BRF-LINKER
User Manual

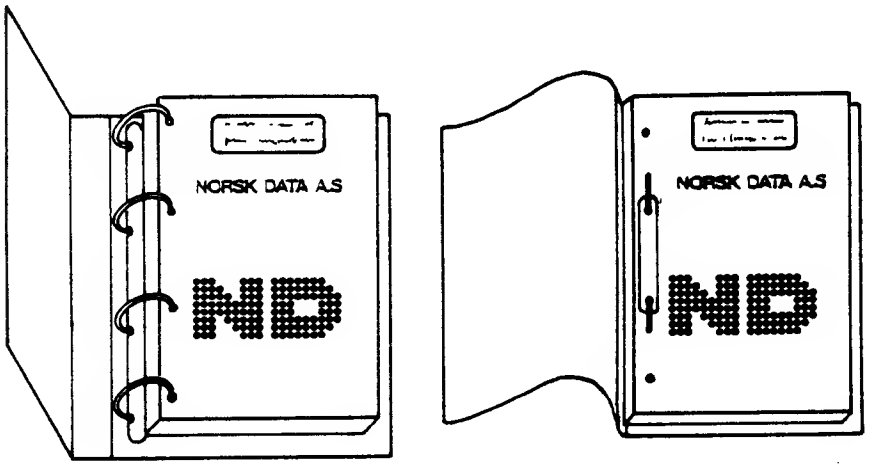
ND-60.196.01

NOTICE

This manual is in loose-leaf form for ease of updating. Old pages may be removed and new pages easily inserted if the manual is revised.

The loose-leaf form also allows you to place the manual in a ring binder (A) for greater protection and convenience of use. Ring binders with 4 rings corresponding to the holes in the manual may be ordered in two widths, 30 mm and 40 mm. Use the order form below.

The manual may also be placed in a plastic cover (B). This cover is more suitable for manuals of less than 100 pages than for large manuals. Plastic covers may also be ordered below.



A: Ring Binder B: Plastic Cover

Please send your order to the local ND office or (in Norway) to:

Norsk Data A.S
Graphic Center
P.O. Box 25, Bogerud
0621 Oslo 6, Norway

ORDER FORM

I would like to order

..... Ring Binders, 30 mm, at nkr 20,- per binder

..... Ring Binders, 40 mm, at nkr 25,- per binder

..... Plastic Covers at nkr 10,- per cover

Name .....
Company .....
Address .....
.....
City .....

PRINTING RECORD

Printing	Notes
08/84	VERSION 01

BRF-LINKER User Manual
Publ.No. ND-60.196.01
August 1984

NORSK DATA A.S
P.O. Box 25, Bogerud
0621 Oslo 6, Norway

Manuals can be updated in two ways, new versions and revisions. New Iversions consist of a complete new manual which replaces the old manual. New versions incorporate all revisions since the previous version. Revisions consist of one or more single pages to be merged into the manual by the user, each revised page being listed on the new printing record sent out with the revision. The old printing record should be replaced by the new one.

New versions and revisions are announced in the ND Bulletin and can be ordered as described below.

The reader's comments form at the back of this manual can be used both to report errors in the manual and to give an evaluation of the manual. Both detailed and general comments are welcome.

These forms and comments should be sent to:

Documentation Department
Norsk Data A.S
P.O. Box 25, Bogerud
0621 Oslo 6, Norway

Requests for documentation should be sent to the local ND office or (in Norway) to:

Graphic Center
Norsk Data A.S
P.O. Box 25, Bogerud
0621 Oslo 6, Norway

Preface:

THE PRODUCT

This manual describes the BRF-Linker, ND-10721A, running under SINTRAN III.

The BRF-Linker is used to read Binary Relocatable Format (or "BRF") output from the MAC assembler and from the ND compilers (FORTRAN, COBOL, PLANC, BASIC, PASCAL, etc.). It will then link this output into a program file and make it executable.

Note that the Multisegment Load feature described in chapter 3 is only available under SINTRAN III version I or later versions. It is therefore not available on the NORD-10.

THE READER

This manual is written for programmers using the BRF-Linker to load and link programs to be run in the time-sharing mode. (For loading of real time programs, see the Real Time Loader manual, ND-60.051.)

PREREQUISITE KNOWLEDGE

No previous knowledge of the BRF-Linker is assumed in this manual. However, some basic knowledge of SINTRAN III commands and of the principles and commands for compilation is recommended.

THE MANUAL

This manual describes the basic commands for loading in chapter 1. Overlay loading is described in chapter 2, and multisegment loading in chapter 3. In chapter 4, can be found some commands for inspection and modification, and in chapter 5, the commands for editing are explained. A detailed description of the Binary Relocatable Format is found in chapter 6.

A summary of the commands is given in appendix A and a summary of the various error messages in appendix B. Furthermore, all commands and error messages are included in the index.

RELATED MANUALS

SINTRAN III Reference Manual	ND-60.128
Real Time Loader	ND-60.051

TABLE OF CONTENTS

Section		Page
1	**THE FUNCTIONS OF THE BRF-LINKER**	**1**
1.1	Command Formats	1
1.2	Loading	2
1.3	Normal Mode Loading	4
1.4	Example: Compiling, Loading and Running a Program	6
1.5	Inspecting and Changing the Symbol Table	6
1.6	Two-bank Systems Versus One-bank Systems	9
1.7	Program Information Commands	10
1.8	Miscellaneous Commands	12
2	**THE OVERLAY SYSTEM**	**13**
2.1	The Multilevel Overlay System	13
2.2	Designing an Overlay Structure	15
2.3	Special Commands for Overlay Loading	16
2.4	Example: Creating an Overlay System	17
3	**THE MULTISEGMENT SYSTEM**	**21**
3.1	The SINTRAN III Segment Files	21
3.2	Programming Considerations Using Multisegment Linking	22
3.3	Organization of a Multisegment Program System	22
3.4	Multisegment Linking Commands	24
3.4.1	Special BRF-Linker Commands for Multisegment Linking	24
3.4.2	SINTRAN III Commands for Multisegment Programs	26
3.5	Example: Linking a Segmented Program Structure	27
4	**PROGRAM INSPECTION COMMANDS**	**35**
5	**EDITING COMMANDS**	**37**
5.1	Basic Symbol Handling	37
5.2	Commands for Updating	38
5.3	Additional Symbol Commands	38
5.4	Other Functions	39
6	**THE BINARY RELOCATABLE FORMAT**	**41**
6.1	The BRF Structure	42
6.2	Relocation of Internal Addresses	43
6.3	Program Units	43
6.4	Separate Compilation	44

Section Page

6.5 Linking of Program Units 44
6.6 FORTRAN COMMON Blocks 45
6.7 Fix-up Facilities 46
6.8 Checksum 46
6.9 Description of the BRF Control Numbers 46

Appendix

A COMMAND SUMMARY *53*

B ERROR MESSAGES *59*

Index 63

1. THE FUNCTIONS OF THE BRF-LINKER

The BRF-Linker is a subsystem which is able to convert the output from language processors (compilers and assemblers) into executable programs that can run under SINTRAN III. The object files created by the language subsystems are in Binary Relocatable Format (described in detail in chapter 6), otherwise known as BRF.

The BRF-Linker maintains a symbol table in which all defined intermodule references, symbols, and labels appear together with their addresses. If the address of a symbol has not been defined before being used, the symbol entry in the table is marked as undefined. All symbols must be defined before the program can be executed.

1.1 Command Formats

BRF-Linker is started by typing its name to SINTRAN III:

```
@BRF-LINKER
```

Whenever BRF-Linker is ready to process a user command, it will type out the command prompt:

```
Brl:
```

BRF-Linker commands follow the same rules as SINTRAN III commands:

- All commands consist of a command name, followed by zero or more parameters.
- A space or comma may be used as a separator between the command name and the parameters, or between two parameters.
- Command names and parameters may be abbreviated as long as the abbreviation is unique.
- A missing parameter is indicated by typing two consecutive commas. Default values will be used for any missing parameters.
- Some parameters are termed optional. These parameters may be specified in the command, but if left out the BRF-Linker will not ask for them, it will just use the default value.
- A carriage return may be used anywhere in the command string. The BRF-Linker will ask for any parameters, except optional ones, that were not specified before the carriage return.
- Numerical parameters may be given in octal or decimal mode. The default is octal mode. A decimal number may be specified by a trailing D, an octal number by a trailing B. Signed numbers may be used.

- All control characters available for editing SINTRAN III commands can also be used to edit commands to the BRF-Linker.

Thus, in the commands:

```
Brl: LOAD FILE-1,FILE-2,FILE-3
Brl: EXIT
```

the words LOAD and EXIT are command names. The EXIT command has no parameters, whereas the LOAD command has the three parameters FILE-1, FILE-2 and FILE-3, separated by commas.

In the command format definitions the parameters are specified in angular brackets (< ... >). Optional parts of the command are enclosed in square brackets ([...]). A sequence of full stops following a parameter means that the parameter may be repeated any number of times.

Thus, the command definition:

```
Brl: LOAD <file name>[,<file name>...]
```

means that the LOAD command takes as parameters any number of file names, of which all but the first are optional (that is, only the first one will be asked for if not specified).

Throughout this manual, two different terms are used to denote quantities of memory, in addition to the usual terms **bit** and **byte**. The symbols are: **word** which denotes one 16-bit ND-100 word, and **page** which is an ND synonym for 1024 16-bit words.

1.2 Loading

The loading operation consists of fetching relocatable program units produced by language processors (compilers and assemblers), placing them in the correct place within the address space, linking together the references between the different units and, finally, writing the completed program out to a program file.

The relocatable program units contain information that makes it possible to place (locate) them anywhere within the address space. This means that the different units may be placed in the address space in any sequence. When BRF-Linker has put a program unit in the correct position, it must go through the program unit and change all addresses that depend on where the unit is placed.

The final program resulting from the loading is bound to the logical addresses where it was placed by BRF-Linker. It is therefore referred to as an absolute program. It may also be called an executable program or a subsystem.

During loading, the BRF-Linker can operate in different modes:

1) **Normal mode**:
   The loading is done onto a file of type :PROG. This is the "normal" way of loading a program. Programs must fit into the ordinary 64-page (one-bank) or 128-page (two-bank) address space.

2) **Overlay mode**:
   When the program is too large to fit into 128 pages, the overlay mode may be used to enable different parts of the program to be run alternately in the same address space.

3) **Multisegment mode**:
   Used to prepare programs which occupy several SINTRAN III segments. It makes it possible to use programs extending beyond the normal 128-page boundaries, and also to improve execution times by avoiding reading from a :PROG file when the program is started. It can NOT be used with one-bank programs.

No symbolic source code modification is necessary in order to switch from one of these modes of loading to another.

There are some significant differences between multisegment linking and overlay linking:

1) The Symbolic Debugger can be used with overlays, but is not available in the multisegment mode.

2) The finished overlay system uses the monitor call RFILE to read code and data during execution of the loaded program. Multisegment linking uses the demand paging facilities with named two-bank segments that is available in SINTRAN III version I and later versions.

3) It takes about 5 milliseconds to switch between segments in the multisegment mode, while it takes at least 50 milliseconds before execution of a new overlay can start after it has been called.

4) In multisegment loading, segments can be built during several loading sessions. When building overlay systems, the entire system must be built in a single BRF-Linker session.

5) Multisegment loading requires the use of special SINTRAN III commands which are only available to user SYSTEM. Overlay loading may be done by any user.

6) Subroutine calls within an overlay structure are restricted in that one routine may call another routine only if both are in memory at the same time. Thus, the user must be careful in organizing the overlay structure. No restrictions on routine calls apply to multisegment systems.

7) In multisegment systems, care must be taken with data area layout to avoid data from one segment being overwritten by data from another segment.

1.3 Normal Mode Loading

An executable, or absolute program is always built on a file. The file is specified using the command

Brl: PROGRAM-FILE <file name>

where <file name> is the name of the file onto which the program is linked and loaded. The default file type is :PROG. If the file does not already exist, the user should instruct the BRF-Linker to make a new file by enclosing the file name in double quotes, thus:

Brl: PROGRAM-FILE "<file name>"

PROGRAM-FILE should be the first command given after the BRF-Linker has been started.

The BRF-Linker can load BRF-units from one or more files. The loading is initiated by the command:

Brl: LOAD <file name>[,<file name>...]

where <file-name> is the name of a file the BRF units should be loaded from. The default file type is :BRF.

When loading from a file, all routines on that file will normally be loaded. Any or all routines on the file may, however, have been compiled in the so-called library-mode. Such routines will only be loaded if they are called from a previously loaded routine, otherwise they will be ignored.

Debug information on BRF files can be included or ignored throughout the loading process by the command:

Brl: DEBUG-MODE <ON/OFF>

Default is ON - debug information will be included.

Program units from library files (compiled with the "LIBRARY-MODE" ON), can be loaded without being referred to from units already loaded by using the command:

Brl: LIBRARY-MODE <ON/OFF>

The default value for this command is ON, library units will only be loaded if referenced.

If the program is in a high-level language, the runtime system routines for that language must also be loaded. These routines are found on files with names like:

xxxxxxx-1BANK:BRF or xxxxxxx-2BANK:BRF

where xxxxxxx is the name of the programming language, for example:

FORTRAN-1BANK:BRF or FORTRAN-2BANK:BRF

Use the 1BANK or 2BANK version of the runtime system depending on whether the program is a one-bank or a two-bank program.

To leave the BRF-Linker and return to SINTRAN III, give the command:

Brl: EXIT

The BRF-Linker will then close the program file specified, thereby making it ready for execution from SINTRAN III, and return you to SINTRAN III.

The program can now be started from SINTRAN III by giving a RECOVER command with the program file name as parameter. For example, if the user has loaded executable code onto the file EXAMPLE:PROG, then the program could be started by typing the command:

@RECOVER EXAMPLE

As long as there is no conflict between the program file name and any SINTRAN III command names we may (and usually do) leave out the word RECOVER, so we would just type:

@EXAMPLE

If we want to debug the program we may instead type the command:

@DEBUG EXAMPLE

which will start up the program under control of the Symbolic Debugger.

If we want to run the loaded program immediately, we could instead exit from the BRF-Linker with the command:

Brl: RUN

This command performs an exit from the BRF-Linker and then starts execution of the program file opened with the **PROGRAM-FILE** command at the beginning of the loading session.

Note that the BRF-Linker cannot load programs directly to memory. Hence, a program file must have been specified in order to use the **RUN** command.

1.4 Example: Compiling, Loading and Running a Program

```
@FORTRAN-100
ND-100/NORD-10 ANSI 77 FORTRAN COMPILER - 203053D
FTN: COMPILE TESTP:SYMB,TERMINAL,"TESTP:BRF"

ND-100/NORD-10 ANSI 77 FORTRAN COMPILER - 203053D
SOURCE FILE:   TESTP:SYMB

    1*            PROGRAM TESTP
    2*            WRITE(1,*) 'THIS IS A TEST PROGRAM'
    3*            END

- CPU TIME USED: 0.8 SECONDS.  3 LINES COMPILED.
- NO MESSAGES
- PROGRAM SIZE=69 COMMON SIZE=0
FTN: EXIT

@BRF-LINKER
- BRF Linker -  JULY 3, 1984
Brl: PROGRAM-FILE "TESTP"
Brl: LOAD TESTP,FORTRAN-1BANK
FREE: P 000105-177777
FREE: P 035043-177777
Brl: EXIT

@TESTP

THIS IS A TEST PROGRAM
```

1.5 Inspecting and Changing the Symbol Table

Procedure names, variable names, labels and so on which are defined and needed in the user's program are known as symbols. Symbols may be up to seven characters long. In order to make a loaded and linked program function properly, the BRF-Linker must fill in the correct symbol address wherever there is a reference to a symbol. By symbol address is meant the address of the word or words in memory that are associated with the symbol. Furthermore, the BRF-Linker will keep track of the places where symbols are referred to, but not yet defined and given addresses. Thus, it is able to fill in the necessary information about these addresses when the symbols get defined.

To this end the BRF-Linker keeps a list of all symbols encountered during linking. This list is known as the **symbol table**. The symbol table may be inspected and manipulated by the user during loading.

The symbol table is built by the BRF-Linker from the symbols it encounters in the BRF files. It contains a list of the symbols and the addresses in the computer's memory they will occupy when the program is run. Whenever a definition of a symbol is found in a input file, the value of the current load address is stored as the address of the symbol. The symbol is then known as a **defined symbol**.

If a referenced symbol has not yet been defined, it is stored in the symbol table as an **undefined symbol**. It is then expected to be defined later. For instance, this will normally be the case with symbols representing calls to external procedures which have not yet been loaded.

In the case of programs loaded in the two-bank mode, the load address is to a location in the program bank if it is a procedure name or a label and to a location in the data bank if it is a variable name. In the one-bank case, all references are to the same bank.

To list all symbols in the symbol table, give the command:

Brl: LIST-ENTRIES-DEFINED ,,,

All undefined symbols in the program can be listed by giving the command:

Brl: LIST-ENTRIES-UNDEFINED ,,,

Together with each symbol name will be listed the last address where the symbol was referenced.

The output from the LIST-ENTRIES-DEFINED and LIST-ENTRIES-UNDEFINED commands may be switched to another output device by giving the command:

Brl: OUTPUT-FILE <file name>

where <file name> is the name of the new output file. The default file type is :SYMB. The output device may be reset to the terminal by giving the OUTPUT-FILE command with an empty file name:

Brl: OUTPUT-FILE ,,,

To create a new symbol in the symbol table, use one of the commands:

Brl: DEFINE <symbol>,<address>,<P/D>

or

Brl: DEFINE <symbol>,<symbol>,<P/D>

In the first format, the name <symbol> will be defined as referencing the word given in <address> and in the bank specified in the parameter <P/D>. P specifies a word in the program bank and D a word in the data bank. If the P/D parameter is omitted, the default is the program bank.

In the second format, the first symbol is defined as referencing the same word as the second symbol, which must be defined previously. Beware that if the already defined symbol (the second <symbol> parameter) is located in the data bank (P/D parameter set to D), the P/D parameter must be set to D for the new symbol too, otherwise the BRF-Linker will output an error message indicating a reference to an undefined symbol.

We can find which word an entry refers to by typing:

Brl: DEFINE <symbol>,?,<P/D>

The BRF-Linker then writes the octal address of the symbol on the terminal.

In order to load the program at an address which differs from the current address, use the command format:

Brl: DEFINE <#PCLC/#DCLC>,<address>

The parameter <#PCLC/#DCLC> refers to the current location counter in the program bank (#PCLC) or data bank (#DCLC). Subsequent loading will then be performed from the specified address. This command will also set the BRF-Linker in the specified mode (#PCLC for Program mode or #DCLC for Data mode).

The address of an entry in the symbol table may be entered into a memory location by the command:

Brl: REFERENCE <symbol>,<address>,<P/D>

It doesn't matter if the referenced entry is present in the table or not, as the correct address will be filled in when the symbol value is defined. The **REFERENCE** command creates an 'undefined' symbol if the symbol is not already in the table, and the BRF-Linker expects it to be defined later.

An entry is deleted from the symbol table by:

Brl: REMOVE <symbol>,<P/D>

Symbol names may be renamed by the command:

Brl: RENAME <old symbol>,<new symbol>

To set the restart address of the program file specified in the **PROGRAM-FILE** command, use one of the command formats:

Brl: RESTART <address>

or

Brl: RESTART <symbol>

If <symbol> is used, then <symbol> must be a defined table entry referring to the program bank. The default restart address will be equal to the main start address.

1.6 Two-bank Systems Versus One-bank Systems

To overcome address space constraints in the ND-100, a two-bank system can be utilized if the compiler (PLANC, COBOL, FORTRAN, PASCAL) is capable of generating separate output for the program code and the data part. The address space for each program is limited to 64 pages. A two-bank program uses a separate address space for code and data, thus making it possible to have 64 pages of program code and 64 pages of data.

Since the ND-100 is capable of addressing data by using an alternative page table, programs may, in principle, consist of 64 pages of program code and 64 pages of data. Programs where code and data are separated in this way are called two-bank programs, whereas programs whose code and data share a single address space of 64 pages, are called one-bank programs.

Two-bank object programs may be generated by an option in the various compilers and can be loaded by BRF-Linker. The following should be noted:

- Two-bank programs must be linked with the two-bank version of the appropriate runtime/library system, for example PLANC-2BANK, FORTRAN-2BANK, COBOL-2BANK, etc.

- Care must be taken when linking assembly or NPL routines with two-bank systems.

- One-bank and two-bank programs may not be mixed.

- The code parts of the two-bank systems are, in principle, completely read-only.

- Overlay tree structures are still available, and both the code and data parts are brought in when a link is required.

Two BRF control numbers, PMO and DMO, are used to put the BRF-Linker into program or data mode (see chapter 6).

Programs compiled in two-bank mode are by default loaded into two banks of 64 pages each. In this case, the program executes with all accesses to the data bank via the alternate page table.

All loader commands (DEFINE, REFERENCE, REMOVE) will apply to either the program code or the data bank according to what is specified in the mode (P/D) parameter in the commands.

1.7 Program Information Commands

The commands described in this section can be used independently of the other BRF-Linker commands, and have no effect on the program being loaded. They can even be used when no **PROGRAM-FILE** command has been given.

Brl: PROGRAM-INFORMATION <file name>
[,<Dump Link Information?YES/NO>,<output file>]

The command lists the information block of a program file. The default file type is :PROG.

It will print out the following information: start and restart address, lower and upper bounds for: program, data and debug information.

If the program is an overlay system or a multisegment system, the BRF-linker will also print the file name specified (in PROGRAM-FILE command) when this program file was loaded.

If the file contains overlays, it will also print overlay information.

For multisegment program files, it will print out lower and upper bounds for link information, and it will ask whether link information shall be dumped (the default answer is 'No'). If link information is to be dumped, it will be dumped on the specified output file. The default output file is TERMINAL and the default output file type is :SYMB.

As an example, let us inspect the simple program we compiled and loaded in section 1.4.

```
@BRF-LINKER
- BRF Linker -  JULY 3, 1984

Brl: PROGRAM-INFORMATION TESTP,,,
Start, Restart  : 000011B - 000011B
Program         : 000000B - 035042B
Data            : 177777B - 000000B
Debug           : 000000B - 000000B

Brl: EXIT
```

The program file name specified in the **PROGRAM-FILE** command when the program was loaded, can be changed by the command:

Brl: PATCH-PROGFILE-NAME <file name>,<new name>

The file name is output to the program file in two-bank programs and in overlay programs. This command will locate the file name on the program file and write the <new name> instead. It will inform you if an overlaid file name is found. The SINTRAN III file is not renamed. The maximum number of characters in the file name is 15.

The usefulness of this command stems from the fact that in two-bank programs to be run under SINTRAN III version H or earlier versions, and in overlay programs, the program file is opened according to the name written on the program file itself. If a program file is renamed by using the SINTRAN III **RENAME-FILE** command, the program name written on the file will not be changed. Such changes can be effected with the **PATCH-PROGFILE-NAME** command, or by using the **COPY-PROGFILE** command described below.

Some difficulties may also be caused if execution of two-bank programs owned by another user is attempted under SINTRAN III version H or earlier versions. In this case, the file name written on the program file does not contain information about the owner or directory. Attempts to execute the program will therefore not be successful. Such difficulties can also be overcome by using the **PATCH-PROGFILE-NAME** command. Beware however, that the file name is still limited to a maximum of 15 characters.

Brl: COPY-PROGFILE <source file>,<destination file>
[,<Include Debug?YES/NO>]
[,<Include Link Information?YES/NO>]

This command will copy a program file from <source file> to <destination file>. The default file type is :PROG. If the source file includes debug information, the BRF-Linker will ask whether debug information is to be included or not; thus, the command can be used to strip away debug information if you answer NO. Default is NO debug information copied.

For multisegment files, the BRF-Linker will ask whether link information should be included. The default is NO link information included. If the link information is not included, the program file can no longer be linked to any other program files.

If the source file is overlaid or is a two-bank program, this command will perform a **PATCH-PROGFILE-NAME** command using <destination file> as the new file name.

The BRF-Linker will print out information about the pages copied as shown in this example (our simple little program again).

```
@BRF-LINKER
- BRF Linker -  JULY 3, 1984

Brl: COPY-PROGFILE TESTP,"TESTX",,,
Total no of pages:17B First page:0B Last page:16B Bank no:0 Program
Brl: EXIT
```

1.8 Miscellaneous Commands

The command:

Brl: HELP [<command>]

lists all available commands matching the abbreviation <command>. If no command is specified, all BRF-Linker commands will be listed.

2. THE OVERLAY SYSTEM

Sometimes a large program cannot be run because it is too big to fit into the address space of 64 pages (or 64 pages for the program and 64 pages for data). One commonly used solution is to divide the program into reasonably small parts which can be run one at a time, and in such a way that one part (or subroutine) can use the space freed when another routine has finished. Thus the program will only need the space for those routines that have to be in memory at the same time.

The sets of different routines to be loaded one at a time are called **overlays** or **links** and the process of loading an overlay to replace an existing set of routines is called overlaying these routines.

Building overlays with the BRF-Linker is a convenient way of bypassing the problem of large programs not being able to fit into the address space because:

- Programs built as overlay systems do not need source code modification.

- The Symbolic Debugger is available for overlays.

An overlay structure cannot be made into a reentrant subsystem.

2.1 The Multilevel Overlay System

In order to use the overlay capability on the ND-100, the user must understand how his program operates and the relationship between the modules within it. He should organize his overlay structure (described below) so as to retain in memory the links containing commonly used routines and place the infrequently used routines in links which can overlay one another. For example, a special error recovery routine would only need to be brought into memory when the corresponding error occurred. Each link should be a collection of functionally related modules and be as self-contained as possible, calling other links as infrequently as possible. In particular, references to links which would overlay other links should be kept to a minimum.

A tree structure, called an **overlay structure**, can be used to illustrate the dependencies among the overlay links. In a tree structure, each link has only one immediate ancestor, but it may have more than one immediate descendent. The link containing the required parts of the program and which must always be in memory during execution is called the **root link**. Since the root link receives control at the start of execution, it does not have an ancestor. The remaining links branch away from the root link and are structured according to their interdependencies.

Links which do not have to be in memory at the same time are termed **independent links** whereas links which must be in memory at the same time are termed **dependent links**. For example, two modules which do not reference each other or pass data directly to each other, are independent links. When such links are no longer required in memory,

they can be overlaid by other links which are brought in. On the other hand, a link must have all the links upon which it depends in memory at the same time and cannot therefore overlay them. Every link is dependent on its ancestor, and consequently, on the root link.

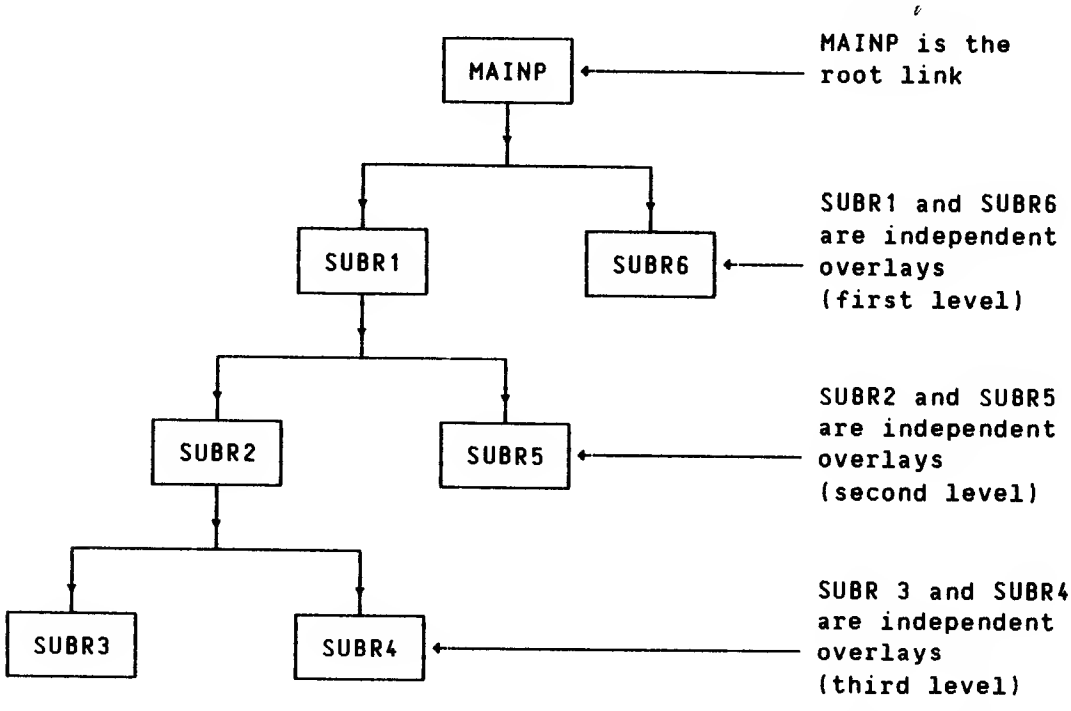
As an illustration, assume we have a program consisting of a main program MAINP and six subroutines SUBR1, SUBR2, SUBR3, SUBR4, SUBR5 and SUBR6. The subroutines are related as follows:

1) SUBR1 and SUBR6 are called directly from MAINP and are independent of each other.

2) SUBR2 and SUBR5 are called directly from SUBR1 and are independent of each other.

3) SUBR3 and SUBR4 are called directly from SUBR2 and are also independent of each other.

The following tree structure illustrates the subroutine dependencies:



SUBR4 depends on SUBR1 and SUBR2 so they must be in memory when in order to execute SUBR4. The chain of links which a link depends on is referred to as the **path** of the link. The action of bringing a link into memory is termed **path loading** and the chain of links branching away from a link is known as the **extended path** of that link. In the previous example, the path of SUBR4 is MAINP, SUBR1, and SUBR2. There are three extended paths of SUBR1:

1) SUBR2, SUBR3
2) SUBR2, SUBR4
3) SUBR5

A link may communicate with other links that lie in its own path or one of its extended paths. The communication is through references to global symbols. A reference from the current link to a global symbol in another link in the path is called a **backward reference**, while a reference from the current link to a global symbol in another link on one of its extended paths is called a **forward reference**. Since all links on the path of the current link must be in memory, a backward reference does not cause any links to be brought into memory. With a forward reference, however, the referenced link may not be in memory. It must then be fetched, possibly overlaying a link already there.

2.2 Designing an Overlay Structure

The first step to be taken when designing an overlay structure is to draw a diagram showing the functional relationships among the modules within the program. The tree begins with the root link which contains the main program and remains in memory throughout execution. The remainder of the program is contained in the overlay links.

The user should remember several points when drawing his overlay structure:

1) References that will overlay existing links should be minimized.

2) Independent links cannot reference each other; communication is by way of a common link.

3) As a general rule, calls to routines on other links should be forward references, while returns from routines should be backward references.

4) If data is modified during execution, the modification is destroyed once the link is overlaid. Therefore, if data required by another link is modified, then the data must be returned to this other link before the link containing the changed data is overlaid.

5) When a link is to be overlaid, no addresses or references to it should remain.

6) Modules, routines or data areas used by several links should be explicitly loaded into a link that is common to all links using these modules or data areas. For example, a FORTRAN COMMON data area should be in a link in the path of all links referencing it. Moreover, COMMON should be positioned in such a way that it never gets re-initialized after the first call. In other programming languages using the distinction between local and global data, similar considerations must be done for the data which are global to several link paths.

7) The Symbolic Debugger should be used with some care on overlays. Debugger commands affecting program/data in an overlay should not be given until a breakpoint is reached on that overlay. Moreover, these commands are in effect only while the overlay resides in memory. In other words, overlays are always brought into memory fully initialized.

Tree-structured overlay systems can be several levels deep. The amount of memory required to run an overlay system is at least the amount needed for the path using the greatest amount of space. This is not the minimum requirement, however, since special tables must be included when a program is divided into links.

The root link and the COMMON areas defined within it reside in memory throughout the entire execution, while the overlays and the COMMON areas defined within them reside on a random read-only file. This file is specified with the **PROGRAM-FILE** command.

2.3 Special Commands for Overlay Loading

Overlay structures are loaded using the same BRF-Linker commands as for normal loading. However, we also need to specify that we are loading a new link in the overlay structure. This is done by the command:

Brl: OVERLAY <level>,<entry name 1>[,...,<entry name n>]

This command specifies that a new overlay link is to be generated. The parameter <level> is the overlay level, and <entry name 1> to <entry name n> give the names of the subprograms that may be called from the previous level. After this command has been given, the specified subprograms can be loaded from one or more BRF files. It is recommended that the overlay subprograms be kept on a separate BRF file compiled in library mode. In this way, the specified set of subprograms may be selected and put into the overlay independently of the compilation sequence.

The level number in an OVERLAY command must not be more than 1 higher than the level number in the previous OVERLAY command.

The special form:

Brl: OVERLAY 0,,

should be used to indicate the start of the root link. This should be the first command following the PROGRAM-FILE command.

The special form:

Brl: OVERLAY -1,,

will append the last overlaid data part to the previously appended one in 2-bank programs. This permits all data to be placed consecutively with no data overlay. Make sure that no previous data overlays share this area with the current data overlay.

To dump the root link, the COMMON area, and the last overlay link onto the file specified in the **PROGRAM-FILE** command, use either the **EXIT** or the **RUN** commands. If you use the **RUN** command, the execution of the overlay system will start immediately, otherwise the execution of the overlay system must be started by a separate command (RECOVER).

2.4 Example: Creating an Overlay System

```
@FORTRAN-100
ND-100/NORD-10 ANSI 77 FORTRAN COMPILER   203053D

FTN: SEPARATE-DATA ON
FTN: COMPILE MAINP,TERMINAL,"MAINP"

ND-100/NORD-10 ANSI 77 FORTRAN COMPILER   203053D
SOURCE FILE:   MAINP:SYMB

    1*              PROGRAM MAINP
    2*              WRITE (1,*) 'START MAINP'
    3*              CALL SUBR1(1)
    4*              CALL SUBR6(6)
    5*              WRITE (1,*) 'END MAINP'
    6*              END

- CPU TIME USED: 0.8 SECONDS.  6 LINES COMPILED.
- NO MESSAGES
- PROGRAM SIZE=53 DATA SIZE=62 COMMON SIZE=0
FTN: EXIT

@FORTRAN-100
ND-100/NORD-10 ANSI 77 FORTRAN COMPILER   203053D

FTN: SEPARATE-DATA ON
FTN: LIBRARY-MODE ON
FTN: COMPILE SUBR1,TERMINAL,"SUBR1"

ND-100/NORD-10 ANSI 77 FORTRAN COMPILER   203053D
SOURCE FILE:   SUBR1:SYMB

    1*              SUBROUTINE SUBR1(N)
    2*              WRITE (1,*) 'SUBROUTINE ',N, ' CALLED'
    3*              CALL SUBR2(2)
    4*              CALL SUBR5(5)
    5*              END

- CPU TIME USED: 0.7 SECONDS.  5 LINES COMPILED.
- NO MESSAGES
- PROGRAM SIZE=35 DATA SIZE=59 COMMON SIZE=0
FTN: EXIT

@FORTRAN-100
ND-100/NORD-10 ANSI 77 FORTRAN COMPILER   203053D

FTN: SEPARATE-DATA ON
FTN: LIBRARY-MODE ON
FTN: COMPILE SUBR2,TERMINAL,"SUBR2"
```

```
ND-100/NORD-10 ANSI 77 FORTRAN COMPILER   203053D
SOURCE FILE:   SUBR2:SYMB

    1*              SUBROUTINE SUBR2(N)
    2*              WRITE (1,*) 'SUBROUTINE ',N, ' CALLED'
    3*              CALL SUBR3(N+1)
    4*              CALL SUBR3(N+1)
    5*              CALL SUBR4(N+2)
    6*              CALL SUBR4(N+2)
    7*              END

- CPU TIME USED: 0.8 SECONDS.  7 LINES COMPILED.
- NO MESSAGES
- PROGRAM SIZE=55 DATA SIZE=63 COMMON SIZE=0
FTN: EXIT

@FORTRAN-100
ND-100/NORD-10 ANSI 77 FORTRAN COMPILER   203053D

FTN: SEPARATE-DATA ON
FTN: LIBRARY-MODE ON
FTN: COMPILE SUBR,TERMINAL,"SUBR"

ND-100/NORD-10 ANSI 77 FORTRAN COMPILER   203053D
SOURCE FILE:   SUBR:SYMB

    1*              SUBROUTINE SUBR3(N)
    2*              WRITE (1,*) 'SUBROUTINE ',N, ' CALLED'
    3*              END

ND-100/NORD-10 ANSI 77 FORTRAN COMPILER   203053D
SOURCE FILE:   SUBR:SYMB

    4*              SUBROUTINE SUBR4(N)
    5*              WRITE (1,*) 'SUBROUTINE ',N, ' CALLED'
    6*              END

ND-100/NORD-10 ANSI 77 FORTRAN COMPILER   203053D
SOURCE FILE:   SUBR:SYMB

    7*              SUBROUTINE SUBR5(N)
    8*              WRITE (1,*) 'SUBROUTINE ',N, ' CALLED'
    9*              END

ND-100/NORD-10 ANSI 77 FORTRAN COMPILER   203053D
SOURCE FILE:   SUBR:SYMB

   10*              SUBROUTINE SUBR6(N)
   11*              WRITE (1,*) 'SUBROUTINE ',N, ' CALLED'
   12*              END

- CPU TIME USED: 1.9 SECONDS.  12 LINES COMPILED.
- NO MESSAGES
- PROGRAM SIZE=100 DATA SIZE=156 COMMON SIZE=0
FTN: EXIT
```

```
@BRF-LINKER
- BRF Linker -  JULY 3, 1984
Brl: PROGRAM-FILE MAINP
Brl: OVERLAY 0,,
Brl: LOAD MAINP,FORTRAN-2BANK
FREE: P 000065-177777     D 000076-177777
FREE: P 027123-177777     D 007430-177777

Brl: OVERLAY 1,SUBR1
Brl: LOAD SUBR1,FORTRAN-2BANK
FREE: P 027213-177777     D 007622-177777     DEBUG 000004
FREE: P 027213-177777     D 007622-177777     DEBUG 000004

Brl: OVERLAY 2,SUBR2
Brl: LOAD SUBR2,FORTRAN-2BANK
FREE: P 027326-177777     D 007720-177777     DEBUG 000010
FREE: P 027326-177777     D 007720-177777     DEBUG 000010

Brl: OVERLAY 3,SUBR3
Brl: LOAD SUBR,FORTRAN-2BANK
FREE: P 027403-177777     D 007766-177777     DEBUG 000014
FREE: P 027403-177777     D 007766-177777     DEBUG 000014

Brl: OVERLAY 3,SUBR4
OVERLAY COMPLETED. BLOCK NO: 2001    27352-27403/7720-7766
SUBR3....27352 P  *........27403 P
*.........7766 D
Brl: LOAD SUBR,FORTRAN-2BANK
FREE: P 027403-177777     D 007766-177777     DEBUG 000024
FREE: P 027403-177777     D 007766-177777     DEBUG 000024

Brl: OVERLAY 2,SUBR5
OVERLAY COMPLETED. BLOCK NO: 2003    27352-27403/7720-7766
SUBR4....27352 P  *........27403 P
*.........7766 D
OVERLAY COMPLETED. BLOCK NO: 2005    27237-27352/7622-7720
SUBR2....27237 P  *........27352 P
*.........7720 D
Brl: LOAD SUBR,FORTRAN-2BANK
FREE: P 027270-177777     D 007670-177777     DEBUG 000040
FREE: P 027270-177777     D 007670-177777     DEBUG 000040

Brl: OVERLAY 1,SUBR6
OVERLAY COMPLETED. BLOCK NO: 2007    27237-27352/7622-7720
SUBR5....27237 P  *........27270 P
*.........7670 D
OVERLAY COMPLETED. BLOCK NO: 2011    27150-27237/7530-7622
SUBR1....27150 P  *........27237 P
*.........7622 D
Brl: LOAD SUBR,FORTRAN-2BANK
FREE: P 027201-177777     D 007576-177777     DEBUG 000054
FREE: P 027201-177777     D 007576-177777     DEBUG 000054
Brl: EXIT
OVERLAY COMPLETED. BLOCK NO: 2013    27150-27201/7530-7576
SUBR6....27150 P  *........27201 P
*.........7576 D
```

@MAINP

```
START MAINP
SUBROUTINE              1 CALLED
SUBROUTINE              2 CALLED
SUBROUTINE              3 CALLED
SUBROUTINE              3 CALLED
SUBROUTINE              4 CALLED
SUBROUTINE              4 CALLED
SUBROUTINE              5 CALLED
SUBROUTINE              6 CALLED
END MAINP
```

3. THE MULTISEGMENT SYSTEM

The need sometimes arises for programs which are as big and extensive as those built by overlay linking, but which are not organized hierarchically like them. The BRF-Linker allows you to build such programs by using of SINTRAN III's mechanism for handling named reentrant segments.

This mechanism is only available in SINTRAN III version I or later versions. In particular, this means that it is not available on the NORD-10.

3.1 The SINTRAN III Segment Files

In order to be able to create and use the multisegment linking facility, the programmer should grasp certain sides of the SINTRAN III operating system. This is provided in this section, which may be skipped by advanced SINTRAN III users.

The key element in the SINTRAN III virtual memory system is the segment file. This is a large, contiguous file on the system disk. The segment file is divided into contiguous areas called segments. A program to be executed must first be put into a segment on the segment file. The different pages of the program will then be swapped into main memory as they are referenced. When the computer's main memory is full, the least recently used pages will be swapped back to their segments.

For every terminal connected to the computer there is a special segment, called a background segment, reserved on the segment file. When an ordinary program is started from a terminal, it is transferred to the terminal's background segment. From there it will be swapped into the main memory as needed. In this way, when several users are running the same program they will still have separate copies of it.

A program may be either one-bank or two-bank. In a one-bank program, both program code and data are loaded into the same 64-page address space, or bank. In a two-bank program, the program code and data are loaded into two separate 64-page banks, making possible a total program size of up to 128 pages. Two-bank programs are usually compiled with the SEPARATE-DATA option in the compiler turned ON.

A background segment may be either 64 or 128 pages long. If only one-bank programs will be run from a terminal, then a 64-page background segment will suffice. In order to run two-bank programs, however, we need a 128-page background segment.

Heavily used programs may be permanently installed on their own segments in the segment file. Such programs are called reentrant subsystems. Their pages will then be swapped in from their segments instead of from background segments. In this case, the same memory copy of a page will be shared between all users running the program, as long as it is not modified. If a user tries to modify a shared page, he will get his own private copy of the page instead, and this

private copy will be swapped to his background segment. Thus a reentrant subsystem will, during runtime, consist of two different kinds of pages. Some will be unmodified, shared pages from the reentrant segment. The rest will be modified, private pages from the user's background segment.

3.2 Programming Considerations Using Multisegment Linking

The BRF-Linker uses the two-bank named reentrant segments mechanism to make multisegment linking possible. This method of combining many routines on several segments has the advantage that overlays will not have to be read from a file during execution; control just switches from one segment to another instead. Another advantage is that the links need not be organized hierarchically, giving no means of communication between links on the same overlay levels, only along different branches of the overlay tree. Instead the program may switch freely between the various links.

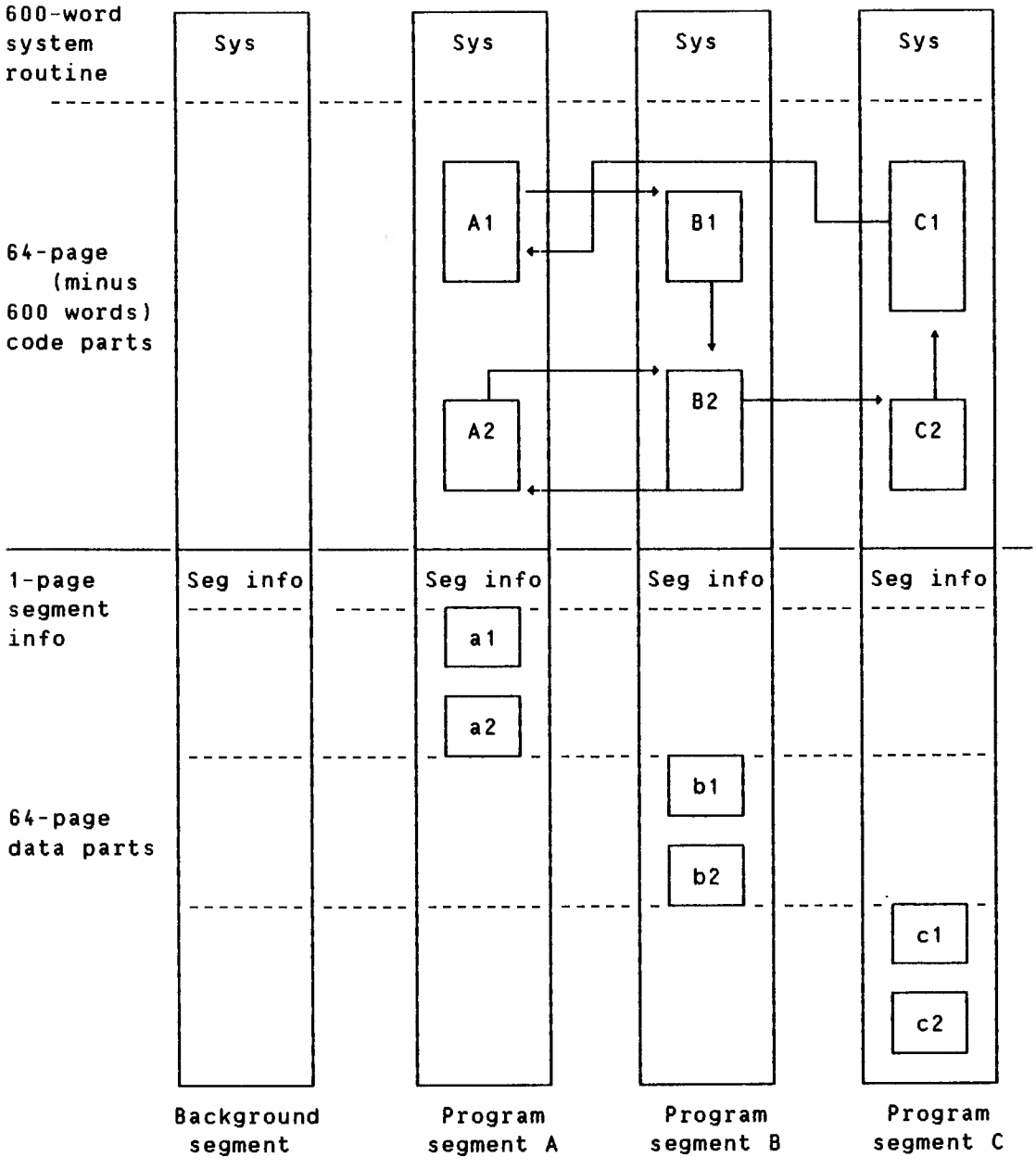
Multisegment linking only works on two-bank programs. Therefore, all routines in a multisegment structure must be compiled with the SEPARATE-DATA option turned ON. Afterwards, the programs linked together in a multisegment structure must be dumped as reentrant segments on the segment file. SINTRAN III commands relating to the administration of segment files are found in the version of the SINTRAN III Reference Manual (ND-60.128) and SINTRAN III System Supervisor (ND-30.003) that pertain to your installation. The commands for dumping programs onto the segment files are privileged, which means that they are only available to the user SYSTEM.

It is not possible to combine multisegment and overlay linking within the same program system.

Data areas which must be globally accessible throughout execution of a multisegment program system must fit into areas of data space which are not used for any other purpose in any segment accessed by that program system. Furthermore, such data areas must be loaded so that they do not overlap. It is, of course, also possible to keep an area global to some subroutines, and to use it for other purposes as soon as these have finished execution. It is not possible, however, to create holes in the data areas; they must be loaded consecutively from the start address for that segment.

3.3 Organization of a Multisegment Program System

The following illustration shows how the multisegment structure is organized on the segments. Even if the drawing shows one particular program structure, the use of segment space is the same here as in every other application of the multisegment, so the information it gives is general.



Three program segments plus the user's background segments are used here. The segments have been named A, B and C during linking, and the subroutines and programs that they contain have been numbered accordingly with capital letters. The data areas used by each program or subroutine are similarly named in small letters. The drawing shows one possible call structure. The program numbered A1 is the root node, and is started by typing its name as response to SINTRAN III's @-prompt. The program A1 calls the subroutine B1, and from then on the calls may be executed as shown by the arrows on the diagram.

It is not necessary to keep data areas as strictly separated as they are in this illustration. If one subprogram and its associated data areas are not needed any more, the data areas may be used freely by other parts of the program.

3.4 Multisegment Linking Commands

To create a multisegment program, some special commands both to the BRF-Linker and to the SINTRAN III operating system are needed. The reason for this is that during linking, the information necessary to link the program parts together is added to the absolute program file (with extension :PROG) that the BRF-Linker creates. This information is used when the different parts of an absolute program are linked together with the LINK-TO command. Dumping of a multisegment program is done by using some of the SINTRAN III commands available to user SYSTEM.

3.4.1 Special BRF-Linker Commands for Multisegment Linking

As mentioned in the previous section, the programs which we want to link into a multisegment system must be transferred from a user file to a named segment in a segment file after loading and linking. During loading, the program file must be specified using a special form of the **PROGRAM-FILE** command:

Brl: PROGRAM-FILE <file name>/<segment name>

The <segment name> is the name of the segment where the reentrant subsystem will be dumped. This name must be used with the SINTRAN III commands necessary to place the linked elements on the segment file. These commands are described in the next section.

The links between the programs on this file and the programs on other files are established with the command:

Brl: LINK-TO <file-1>, ... <file-n>

where each <file-n> is a program file with links to/from the current program file. The current program file is the file specified in the **PROGRAM-FILE** command. Each of the files to be linked must have been loaded as a multisegment program files.

When using the command **LINK-TO**, the BRF-Linker will link the n files so that programs in the n segment pairs can call each other. Entries in the files <file-1>, ... <file-n> are matched with the corresponding entries in the current program file. If these files are now dumped to segment files, routines in the current program file may call routines in the link files <file-1>, ... <file-n> and vice versa.

Please note that this matching does not imply that programs in the files <file-1>, ... <file-n> will be able to call each other. If this is desired, a new linking session is needed to establish these links.

When the relevant information has been written on to the program files, the BRF-Linker will respond by answering:

<entry> LINKED FROM <current file> TO <link file>

<entry> LINKED FROM <link file> TO <current file>.

If the BRF-Linker finds the same data or COMMON area in both the current program file and in a link file, it will output the message:

<entry> DEFINED IN BOTH <link file> AND <current file>.

Note that this may not necessarily constitute an error, but you should check carefully that it is not meant to be the same data or COMMON area.

If output has been redefined to a file by the **OUTPUT-FILE** command, output from the **LINK-TO** command will be written to this file.

The **LINK-TO** command will only initiate the linking. The actual linking process takes place after the EXIT command is given.

The multisegment linking can be used with all programs compiled in the two-bank mode. The total global data space (i.e., data space which is available from all segments) is limited to a maximum of 63 pages, the remaining 1 page is used for segment information. Local data space can be overlapped. If a segment using overlapped data space is entered and another segment has used the same data space, initial data will be used for the segment entered.

When loading a segment, the command:

Brl: DEFINE #DCLC,<address>

should be used to place its private data in a suitable area. Due to the paging system, the data area cannot be divided into smaller parts than 20008 (2000 octal) words. The data of that segment will be placed contiguously from that address. The first page (20008 locations) of the data space is used to store segment information.

External data may be shared between segments simply by linking the program files together. No entry names need to be specified. Data on a linked segment will not be available before that segment has been entered (must have been called from another segment). The data applies until another overlapping segment is activated.

If the **LINK-TO** command is given prior to a **LOAD** command, the defined data entries in the files linked will be regarded as defined in the current program file. The entries will not be defined from any **LOAD** commands following **LINK-TO**, but will be linked from the link files at **EXIT**.

If a **FORTRAN COMMON** area is to be linked from another segment, it is defined by linking the program file where the common area is defined to the current program file. All common areas not defined (by LOAD or by LINK-TO), will be defined when the EXIT command is performed.

When using the multisegment system, the start address is 0 and the restart address is 1 for the programs created.

3.4.2 SINTRAN III Commands for Multisegment Programs

When a program file (with extension :PROG) has been created with multisegment linking information on it, it must be transferred to a segment file.

The following commands do that. They must be performed by the user SYSTEM.

@DUMP-PROGRAM-REENTRANT <subsystem-name>,<file>[,<segment-name>]

which dumps the program file for the main program onto a segment in the segment file, and:

@LOAD-REENTRANT-SEGMENT <file>,<segment-name>.

which creates subprogram segments on the segment file.

These SINTRAN III commands must be given after the linking sessions have been finished and the resulting program files have been created.

The reentrant main program segment is accessible to all users. If it is preferable to have some degree of privacy for a multisegment system, the user can dump only the subprogram segments and keep his main program on a program file (with extension :PROG). The main program will be read into the user's background segment when it is requested, and the background segment will subsequently be used as the main program segment. It will be difficult for unauthorized users to use the subprogram segments without having access to the main program.

The segment file area may need to be cleared before loading. The main program segment is deleted by the SINTRAN III command:

@DELETE-REENTRANT <subsystem-name>

and the other segments by:

@CLEAR-REENTRANT-SEGMENT <segment-name>

If the message:

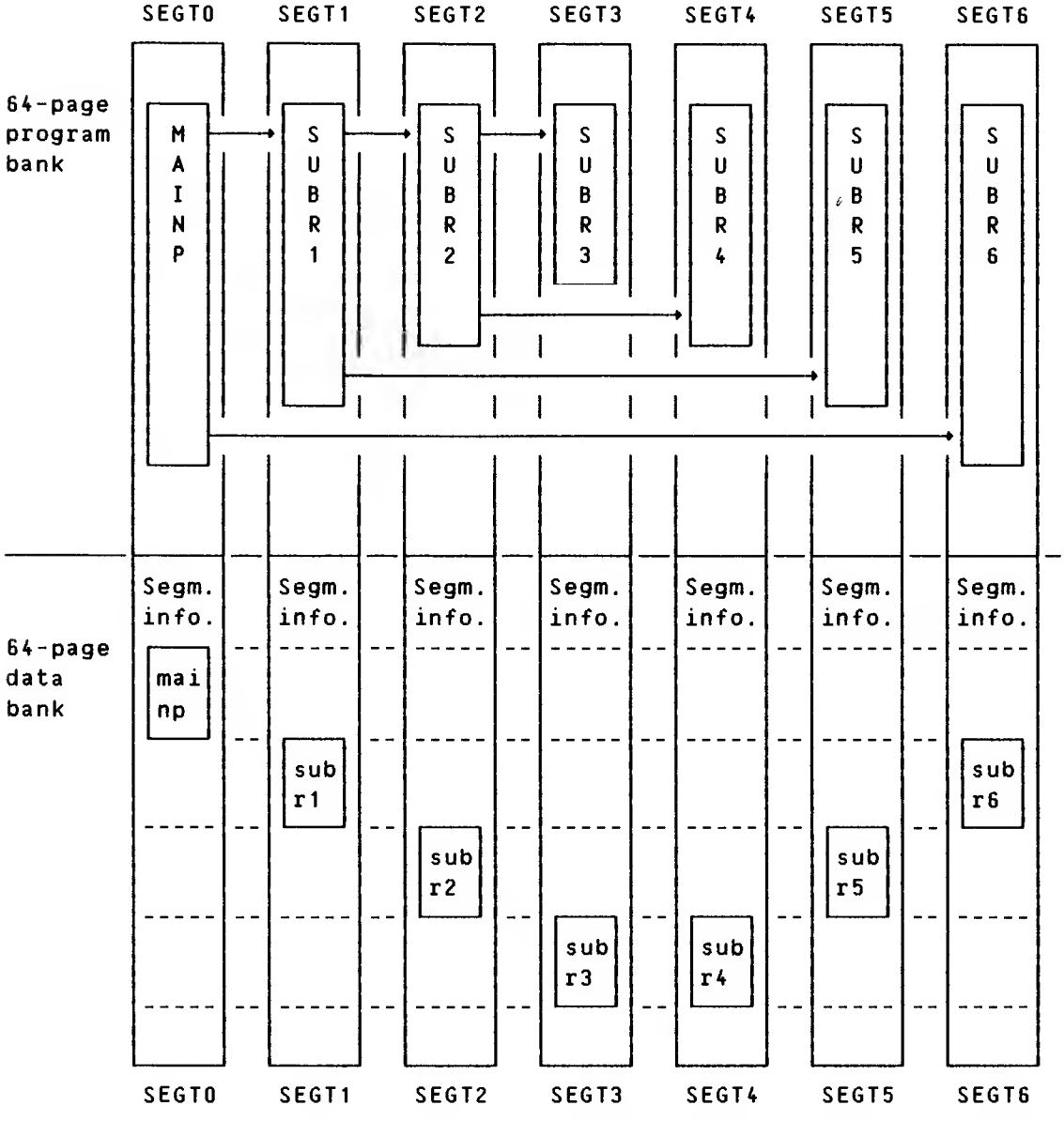
'Segment Number xx is not cleared'

appears, this means the segment is currently in use. The SINTRAN III command CLEAR-REENTRANT-SEGMENT should then be repeated at a later time.

3.5 Example: Linking a Segmented Program Structure

Using the same main program and subroutines as in the example in section 2.4, we now load the main program (MAINP) and its six subroutines (SUBR1, SUBR2, SUBR3, SUBR4, SUBR5 and SUBR6) onto different segments and run it.

The program has the following call structure:



Note that the size of the illustrated subroutines in the program bank does not indicate the actual size, but is chosen in this way to give a better view of the calling sequence.

The following linking session will create this structure:

```
@FORTRAN-100
ND-100/NORD-10 ANSI 77 FORTRAN COMPILER - 203053D
FTN: SEPARATE-DATA ON
FTN: COMPILE MAINP,TERMINAL,"MAINP"

ND-100/NORD-10 ANSI 77 FORTRAN COMPILER - 203053D
SOURCE FILE:   MAINP:SYMB

    1*              PROGRAM MAINP
    2*              WRITE (1,*) 'START MAINP'
    3*              CALL SUBR1(1)
    4*              CALL SUBR6(6)
    5*              WRITE (1,*) 'END MAINP'
    6*              END

- CPU TIME USED: 1.0 SECONDS.  6 LINES COMPILED.
- NO MESSAGES
- PROGRAM SIZE=53 DATA SIZE=64 COMMON SIZE=0
FTN: EXIT

@FORTRAN-100
ND-100/NORD-10 ANSI 77 FORTRAN COMPILER - 203053D
FTN: SEPARATE-DATA ON
FTN: COMPILE SUBR1,TERMINAL,"SUBR1"

ND-100/NORD-10 ANSI 77 FORTRAN COMPILER - 203053D
SOURCE FILE:   SUBR1:SYMB

    1*              SUBROUTINE SUBR1(N)
    2*              WRITE (1,*) 'SUBROUTINE ',N, ' CALLED'
    3*              CALL SUBR2(2)
    4*              CALL SUBR5(5)
    5*              END

- CPU TIME USED: 1.0 SECONDS.  5 LINES COMPILED.
- NO MESSAGES
- PROGRAM SIZE=35 DATA SIZE=59 COMMON SIZE=0
FTN: EXIT

@FORTRAN-100
ND-100/NORD-10 ANSI 77 FORTRAN COMPILER - 203053D
FTN: SEPARATE-DATA ON
FTN: COMPILE SUBR2,TERMINAL,"SUBR2"

ND-100/NORD-10 ANSI 77 FORTRAN COMPILER - 203053D
SOURCE FILE:   SUBR2:SYMB

    1*              SUBROUTINE SUBR2(N)
    2*              WRITE (1,*) 'SUBROUTINE ',N, ' CALLED'
    3*              CALL SUBR3(N+1)
    4*              CALL SUBR3(N+1)
    5*              CALL SUBR4(N+2)
    6*              CALL SUBR4(N+2)
    7*              END
```

```
- CPU TIME USED: 1.0 SECONDS.  7 LINES COMPILED.
- NO MESSAGES
- PROGRAM SIZE=55 DATA SIZE=63 COMMON SIZE=0
FTN: EXIT

@FORTRAN-100
ND-100/NORD-10 ANSI 77 FORTRAN COMPILER - 203053D
FTN: SEPARATE-DATA ON
FTN: COMPILE SUBR3,TERMINAL,"SUBR3"

ND-100/NORD-10 ANSI 77 FORTRAN COMPILER - 203053D
SOURCE FILE:   SUBR3:SYMB

    1*              SUBROUTINE SUBR3(N)
    2*              WRITE (1,*) 'SUBROUTINE ',N, ' CALLED'
    3*              END

- CPU TIME USED: 0.8 SECONDS.  3 LINES COMPILED.
- NO MESSAGES
- PROGRAM SIZE=25 DATA SIZE=39 COMMON SIZE=0
FTN: EXIT

@FORTRAN-100
ND-100/NORD-10 ANSI 77 FORTRAN COMPILER - 203053D
FTN: SEPARATE-DATA ON
FTN: COMPILE SUBR4,TERMINAL,"SUBR4"

ND-100/NORD-10 ANSI 77 FORTRAN COMPILER - 203053D
SOURCE FILE:   SUBR4:SYMB

    1*              SUBROUTINE SUBR4(N)
    2*              WRITE (1,*) 'SUBROUTINE ',N, ' CALLED'
    3*              END

- CPU TIME USED: 1.1 SECONDS.  3 LINES COMPILED.
- NO MESSAGES
- PROGRAM SIZE=25 DATA SIZE=39 COMMON SIZE=0
FTN: EXIT

@FORTRAN-100
ND-100/NORD-10 ANSI 77 FORTRAN COMPILER - 203053D
FTN: SEPARATE-DATA ON
FTN: COMPILE SUBR5,TERMINAL,"SUBR5"

ND-100/NORD-10 ANSI 77 FORTRAN COMPILER - 203053D
SOURCE FILE:   SUBR5:SYMB

    1*              SUBROUTINE SUBR5(N)
    2*              WRITE (1,*) 'SUBROUTINE ',N, ' CALLED'
    3*              END

- CPU TIME USED: 0.8 SECONDS.  3 LINES COMPILED.
- NO MESSAGES
- PROGRAM SIZE=25 DATA SIZE=39 COMMON SIZE=0
FTN: EXIT
```

```
@FORTRAN-100
ND-100/NORD-10 ANSI 77 FORTRAN COMPILER - 203053D
FTN: SEPARATE-DATA ON
FTN: COMPILE SUBR6,TERMINAL,"SUBR6"

ND-100/NORD-10 ANSI 77 FORTRAN COMPILER - 203053D
SOURCE FILE:    SUBR6:SYMB

    1*              SUBROUTINE SUBR6(N)
    2*              WRITE (1,*) 'SUBROUTINE ',N, ' CALLED'
    3*              END

- CPU TIME USED: 0.9 SECONDS.  3 LINES COMPILED.
- NO MESSAGES
- PROGRAM SIZE=25 DATA SIZE=39 COMMON SIZE=0
FTN: EXIT

@BRF-LINKER
- BRF Linker -  JULY 3, 1984
Brl: PROGRAM-FILE "FILE0"/SEGT0
Brl: DEFINE #DCLC,2000
Brl: LOAD MAINP,FORTRAN-2BANK
FREE: P 000665-177777      D 002077-177777
FREE: P 027504-177777      D 010561-177777
Brl: EXIT
SUBR1....27507 U  SUBR6....27523 U

@BRF-LINKER
- BRF Linker -  JULY 3, 1984
Brl: PROGRAM-FILE "FILE1"/SEGT1
Brl: DEFINE #DCLC,12000
Brl: LOAD SUBR1
FREE: P 000643-177777      D 012072-177777
Brl: LINK-TO FILE0
Brl: LOAD FORTRAN-2BANK
FREE: P 027345-177777      D 016474-177777
Brl: EXIT
5PTAB....13267 U  LINKED FROM FILE0 TO FILE1
5EXCINF..12071 U  LINKED FROM FILE0 TO FILE1
5ESTACK..21375 U  LINKED FROM FILE0 TO FILE1
5STACK....1077 U  LINKED FROM FILE0 TO FILE1
5FIO_BL..12067 U  LINKED FROM FILE0 TO FILE1
5USFILB..14655 U  LINKED FROM FILE0 TO FILE1
5CNCT....14656 U  LINKED FROM FILE0 TO FILE1
5ALTREC..22332 U  LINKED FROM FILE0 TO FILE1
SUBR1......600 P  LINKED FROM FILE1 TO FILE0
SUBR2....27350 U  SUBR5....27364 U

@BRF-LINKER
- BRF Linker -  JULY 3, 1984
Brl: PROGRAM-FILE "FILE2"/SEGT2
Brl: DEFINE #DCLC,22000
Brl: LOAD SUBR2
FREE: P 000667-177777      D 022076-177777
Brl: LINK-TO FILE0,FILE1
Brl: LOAD FORTRAN-2BANK
FREE: P 027371-177777      D 026500-177777
```

```
Brl: EXIT
5PTAB....13313 U  LINKED FROM FILE0 TO FILE2
5EXCINF..22075 U  LINKED FROM FILE0 TO FILE2
5ESTACK..21421 U  LINKED FROM FILE0 TO FILE2
5STACK....1123 U  LINKED FROM FILE0 TO FILE2
5FIO_BL..22073 U  LINKED FROM FILE0 TO FILE2
5USFILB..24661 U  LINKED FROM FILE0 TO FILE2
5CNCT....24662 U  LINKED FROM FILE0 TO FILE2
5ALTREC..22356 U  LINKED FROM FILE0 TO FILE2
SUBR2......600 P  LINKED FROM FILE2 TO FILE1
SUBR3....27374 U  SUBR4....27410 U

@BRF-LINKER
- BRF Linker -  JULY 3, 1984
Brl: PROGRAM-FILE "FILE3"/SEGT3
Brl: DEFINE #DCLC,32000
Brl: LOAD SUBR3
FREE: P 000631-177777     D 032046-177777
Brl: LINK-TO FILE0,FILE2
Brl: LOAD FORTRAN-2BANK
FREE: P 027333-177777     D 036450-177777
Brl: EXIT
5PTAB....13255 U  LINKED FROM FILE0 TO FILE3
5EXCINF..32045 U  LINKED FROM FILE0 TO FILE3
5ESTACK..21363 U  LINKED FROM FILE0 TO FILE3
5STACK....1065 U  LINKED FROM FILE0 TO FILE3
5FIO_BL..32043 U  LINKED FROM FILE0 TO FILE3
5USFILB..34631 U  LINKED FROM FILE0 TO FILE3
5CNCT....34632 U  LINKED FROM FILE0 TO FILE3
5ALTREC..22320 U  LINKED FROM FILE0 TO FILE3
SUBR3......600 P  LINKED FROM FILE3 TO FILE2

@BRF-LINKER
- BRF Linker -  JULY 3, 1984
Brl: PROGRAM-FILE "FILE4"/SEGT4
Brl: DEFINE #DCLC,32000
Brl: LOAD SUBR4
FREE: P 000631-177777     D 032046-177777
Brl: LINK-TO FILE0,FILE2
Brl: LOAD FORTRAN-2BANK
FREE: P 027333-177777     D 036450-177777
Brl: EXIT
5PTAB....13255 U  LINKED FROM FILE0 TO FILE4
5EXCINF..32045 U  LINKED FROM FILE0 TO FILE4
5ESTACK..21363 U  LINKED FROM FILE0 TO FILE4
5STACK....1065 U  LINKED FROM FILE0 TO FILE4
5FIO_BL..32043 U  LINKED FROM FILE0 TO FILE4
5USFILB..34631 U  LINKED FROM FILE0 TO FILE4
5CNCT....34632 U  LINKED FROM FILE0 TO FILE4
5ALTREC..22320 U  LINKED FROM FILE0 TO FILE4
SUBR4......600 P  LINKED FROM FILE4 TO FILE2
```

```
@BRF-LINKER
- BRF Linker -  JULY 3, 1984
Brl: PROGRAM-FILE "FILE5"/SEGT5
Brl: DEFINE #DCLC,22000
Brl: LOAD SUBR5
FREE: P 000631-177777     D 022046-177777
Brl: LINK-TO FILE0,FILE1
Brl: LOAD FORTRAN-2BANK
FREE: P 027333-177777     D 026450-177777
Brl: EXIT
5PTAB....13255 U  LINKED FROM FILE0 TO FILE5
5EXCINF..22045 U  LINKED FROM FILE0 TO FILE5
5ESTACK..21363 U  LINKED FROM FILE0 TO FILE5
5STACK....1065 U  LINKED FROM FILE0 TO FILE5
5FIO_BL..22043 U  LINKED FROM FILE0 TO FILE5
5USFILB..24631 U  LINKED FROM FILE0 TO FILE5
5CNCT....24632 U  LINKED FROM FILE0 TO FILE5
5ALTREC..22320 U  LINKED FROM FILE0 TO FILE5
SUBR5......600 P  LINKED FROM FILE5 TO FILE1

@BRF-LINKER
- BRF Linker -  JULY 3, 1984
Brl: PROGRAM-FILE "FILE6"/SEGT6
Brl: DEFINE #DCLC,12000
Brl: LOAD SUBR6
FREE: P 000631-177777     D 012046-177777
Brl: LINK-TO FILE0
Brl: LOAD FORTRAN-2BANK
FREE: P 027333-177777     D 016450-177777
Brl: EXIT
5PTAB....13255 U  LINKED FROM FILE0 TO FILE6
5EXCINF..12045 U  LINKED FROM FILE0 TO FILE6
5ESTACK..21363 U  LINKED FROM FILE0 TO FILE6
5STACK....1065 U  LINKED FROM FILE0 TO FILE6
5FIO_BL..12043 U  LINKED FROM FILE0 TO FILE6
5USFILB..14631 U  LINKED FROM FILE0 TO FILE6
5CNCT....14632 U  LINKED FROM FILE0 TO FILE6
5ALTREC..22320 U  LINKED FROM FILE0 TO FILE6
SUBR6......600 P  LINKED FROM FILE6 TO FILE0
```

Note that in order to get just one copy of the FORTRAN runtime system data tables, the command **LOAD FORTRAN-2BANK** has to be placed **after** the **LINK-TO** command for each subroutine. This is important to remember, because if each subroutine gets its own copy of the runtime system tables, the loaded program may not work.

We can then use the described SINTRAN III commands to load and run these segments. Remember, this loading must be done as user SYSTEM:

```
@DUMP-PROGRAM-REENTRANT BRLDEMO,FILE0,SEGT0

@LOAD-REENTRANT-SEGMENT FILE1,SEGT1

@LOAD-REENTRANT-SEGMENT FILE2,SEGT2

@LOAD-REENTRANT-SEGMENT FILE3,SEGT3

@LOAD-REENTRANT-SEGMENT FILE4,SEGT4

@LOAD-REENTRANT-SEGMENT FILE5,SEGT5

@LOAD-REENTRANT-SEGMENT FILE6,SEGT6

@BRLDEMO

START MAINP
SUBROUTINE                1 CALLED
SUBROUTINE                2 CALLED
SUBROUTINE                3 CALLED
SUBROUTINE                3 CALLED
SUBROUTINE                4 CALLED
SUBROUTINE                4 CALLED
SUBROUTINE                5 CALLED
SUBROUTINE                6 CALLED
END MAINP
```

4. PROGRAM INSPECTION COMMANDS

Sometimes it is necessary to inspect the contents of the loaded program. This can be done by the BRF-Linker, both on executable program files and on a program currently being loaded.

To inspect an existing program file, use the command:

Brl: **PROGRAM-FILE** <file name>,W

When inspecting existing files, the only linker commands that can be used are:

LOOK-AT-PROGRAM, **LOOK-AT-DATA**, **RESTART**, **RUN** and **EXIT**.

Multisegment program files can be inspected and modified, but no segment names may be specified when using the W option. This must be done before they are dumped onto the segment files. Patching of segment files after the dumping has been done is an entirely different topic not covered by this manual.

The commands:

Brl: **LOOK-AT-PROGRAM** <address>

and:

Brl: **LOOK-AT-DATA** <address>

enable the user to inspect and modify program/data locations, both on executable program files and on the results of a loading session before they are written onto such files. The contents of the location will be written on the terminal as a six-digit octal number, as a decimal number and as ASCII characters. If **LOOK-AT-PROGRAM** is used, the symbolic instructions will also be printed.

New contents are entered by typing a new number. The new number may be given in octal or decimal mode. The default is octal mode. A decimal number may be specified by a trailing D, an octal number by a trailing B. Signed numbers may be used.

CR (carriage return) advances to the next address without changing the contents of the item. **EXIT** or . (period) returns control to the BRF-Linker command processor.

5. EDITING COMMANDS

The BRF-Linker can also be used for editing files containing BRF code (output from compilers, the MAC assembler, etc.). The BRF code format is described in chapter 6. The BRF-Linker, used as an editor, can perform such operations as combining files, modifying libraries, etc. Be aware of the following points:

- The BRF-Linker will check all units for syntax errors and checksum errors.
- The default values for the <first unit> and <last unit> parameters are the first and the last BRF units on the file respectively.
- All files used as parameters (except the <output file>) have the default type :BRF.
- The units to be specified in the commands can be identified by any of the names defined by the MAIN or ENTR codes (see chapter 6).

5.1 Basic Symbol Handling

The command:

Brl: LIST-BRF-ENTRIES <file name>,<output file>

will list all defined symbols and their addresses found in <file name> onto the output file. The output will appear in this order: symbol name, address and mode (program or data).

As an example, let us use the BRF-Linker command LIST-ENTRIES to take a look at the SUBR file containing the subroutines SUBR3, SUBR4, SUBR5 and SUBR6 from the overlay example in section 2.4:

```
@BRF-LINKER
- BRF Linker -  JULY 3, 1984
Brl: LIST-BRF-ENTRIES SUBR,,,,
SUBR3.....0 P  SUBR4.....31 P  SUBR5.....62 P
SUBR6.....113 P  *.....144 P
*.....231 D
Brl: EXIT
```

Brl: APPEND-BRF <source file>,<destination file>,<after unit>

The BRF units in the source file will be inserted in the destination file after the unit identified by <after unit>. If no <after unit> is specified, the source file will be appended to the destination file after the last BRF unit in the destination file.

Brl: FETCH-BRF <source file>,<destination file>,<first unit>,
<last unit>

The BRF units in the source file, starting with the <first unit> and including every unit up to and including the <last unit>, will be appended to the destination file following the last BRF unit which appears in it.

Brl: DELETE-BRF <file name>,<first unit>,<last unit>

The specified BRF units will be deleted from the file. The <first unit> will be the first unit deleted, then all the BRF units following it, including <last unit>, will be deleted.

5.2 Commands for Updating

The command:

Brl: REPLACE-BRF <source file>,<destination file>

will replace the BRF units in the destination file with the same name as those in the source file by the BRF units in the source file.

The BRF units in the destination file will have the same relative position within the file after the REPLACE-BRF command as they had before.

BRF units in the source file not found in the destination file will be skipped and a warning message will be issued.

BRF units without symbolic names cannot be replaced.

5.3 Additional Symbol Commands

The command:

Brl: PREPARE-BRF-LIBRARY-FILE <source file>

will set up a BRF unit containing an index table of all the BRF units. The index table is the first BRF unit in the new file. Each element in the index table consists of 5 words: 3 words for the unit name and 2 words for the byte pointer of the unit. Selective loading (search for referenced library units) from a file with an index table will be faster than loading the same file without the index table.

The index table is invalidated by all commands modifying the contents of the BRF file (APPEND-BRF, FETCH-BRF, DELETE-BRF and REPLACE-BRF). The table must be rebuilt if any of these commands are performed.

Brl: INSERT-BRF-MESSAGE <file name>,<before unit>,<message>

This command inserts a message in the BRF file before the specified unit. If the file is prepared with the PREPARE-BRF-LIBRARY-FILE command, the default position is in the front of the index table. The specified message will be printed when the file is loaded. If the file is a library file headed by an index table, any message inserted in front of the index table is printed; all other messages (defined by

this command) are located outside BRF units, and are not written.

Brl: RENAME-BRF <file name>,<old symbol>,<new symbol>

This command changes the name of a symbol in a BRF code file identified by <file name>. The <old symbol> is the current name of the symbol while <new symbol> specifies the new one.

5.4 Other Functions

The command:

Brl: LIST-BRF-CODE <file name>,<first unit>,<last unit>,
<output file>

will list the BRF information regarding the <first unit> and all the other units up to and including <last unit> on the specified source file on the <output file>. The information given is as follows:

- Location counter (octal)
- BRF control number (octal)
- Name of the BRF control number
- All symbolic names (REF, ENTR, LIBR, MAIN, ASF, ADS, etc.)
- Binary information (octal)
- Disassembled (if program code)

As an example, we use the BRF-Linker command **LIST-BRF-CODE** to take a look at the small example program in section 1.4:

```
@BRF-LINKER
- BRF Linker -  JULY 3, 1984
Brl: LIST-BRF-CODE TESTP,,,
     1     17 BEG  *** new BRF - unit ***
     1     32 LONG
     1     11 AFL      11
    12     14 MAIN TESTP
    12     24 LNF       3
    12              171400 SAX 0
    13              135021 JPL I * 21
    14                   0 STZ *
    15      2 LR         0
    16     24 LNF       16
    16                   0 STZ *
    17                   0 STZ *
    20                   0 STZ *
    21                 605 STZ ,B - 173
    22              135013 JPL I * 13
    23               44013 LDA * 13
    24              135013 JPL I * 13
    25               44013 LDA * 13
    26              135013 JPL I * 13
    27              135013 JPL I * 13
    30              170777 SAA - 1
    31              135012 JPL I * 12
    32              124001 JMP * 1
    33              135011 JPL I * 11
```

```
    34     20 REF  5INIT
    35     20 REF  5EXCEPT
    36      2 LR        0
    37     20 REF  5FIO
    40      2 LR        0
    41     20 REF  5DAT
    42     20 REF  5CLS
    43     20 REF  5XCLO
    44     20 REF  5LEAV
    45     24 LNF      14
    45              52110 LDT ,X 110
    46              44523 LDA ,B 123
    47              20111 STD * 111
    50              51440 LDT I ,B 40
    51              40440 MIN ,B 40
    52              52105 LDT ,X 105
    53              51524 LDT I ,B 124
    54              20120 STD * 120
    55              51117 LDT I * 117
    56              43522 MIN ,X I ,B 122
    57              40515 MIN ,B 115
    60               2440 STZ ,X ,B 40
    61      2 LR        45
    62      1 LF        26 STZ * 26
    63      2 LR        60
    64      1 LF         1 STZ * 1
    65      6 AFR       65,40
    65      2 LR        63
    66      2 LR        61
    67     24 LNF        3
    67                   1 STZ * 1
    70                   2 STZ * 2
    71              13000 STT ,X I *
    72      6 AFR       72,36
    72      2 LR        70
    73      2 LR        67
    74      1 LF   177606 BORA 0 DT
    75      6 AFR       75,15
    75     11 AFL        6
   103     20 REF  5FIO_BL
   104      1 LF         0 STZ *
   105     20 REF  5EXCINF
   106      7 ARR      103,77
   106      7 ARR      103,100
   106      6 AFR        0,14
   106      6 AFR        6,16
   106     16 ENTR label 1,0,0
   106     21 END  checksum :  72054
Brl: EXIT
```

6. THE BINARY RELOCATABLE FORMAT

A program is a set of instructions and data which, when executed, will perform an algorithm. A program may be in various forms. It may be written in FORTRAN, assembly code, machine code, etc. But the most important aspect is whether it is bound to a specific location in memory or not. We refer to a program that can be moved to another part of memory as a relocatable program.

Thus, a FORTRAN program and an assembly program (with only symbolic addresses) are relocatable programs, while a program in binary form is generally not relocatable. Consider the following three versions of the same program:

Program ABC written in assembly code		Program ABC written in binary form (placed from location 10)	Program ABC written in binary form (placed from location 20)
ABC,	JMP I *+1	125001	125001
	XYZ	14	24
	157	157	157
	751	751	751
XYZ,	WAIT	151000	151000

The binary program version which is bound to location 10 cannot be moved to location 20 without changes. The machine code is not in relocatable format, since there is no information about which words contain internal addresses that have to be modified depending on the placement of the program.

If the language processors (compilers and assemblers) produced machine code directly, this would cause serious problems for programmers. Since every routine would be fixed in a specific place in memory, any modification that would change the length of any routine would mean that the whole program system would have to be recompiled. Using separately compiled routines (including runtime system routines) or combining routines written in different languages would be difficult or impossible.

For this reason most language processors generate relocatable code. The relocatable code format used on ND-100 computers is called BRF (Binary Relocatable Format). In this format, information about references between the various parts of the program system, such as procedure calls, references to global data, etc., is coded as **symbols**. These symbols are alphanumeric names assigned by the compiler to an instruction or to a data item. The memory locations where these instructions and data items will eventually be placed are selected by the BRF-Linker according to how it places the various program parts in memory.

6.1 The BRF Structure

BRF code is organized in eight-bit bytes and can be stored on any data medium (magnetic tape, disk, etc.). The information contained in the object program may be organized in the following kind of groups:

- **Control information** is held in a **control byte** (which forms the **control number**) and is interpreted as loader commands.
- **Programmed information** is held in two bytes containing a sixteen-bit word and is termed a **P-group**.
- **Symbolic information** is held in four bytes for MAC and NPL, and six bytes for FORTRAN, COBOL, etc. This is termed an **S-group** containing a symbol of one to seven six-bit characters.

For further information see the MAC Interactive Assembly and Debugging System User's Guide (ND-60.096).

BRF code is made up of a sequence of BRF groups. A BRF group can take on one of the following forms:

<control byte>
<control byte><P-group><P-group>
<control byte><S-group>
<control byte><S-group><P-group>

The example program ABC will look like this when broken down columnwise into BRF groups:

Control byte mnemonics	Control byte	P-group
BEG	17	
LF	1	125001
LR	2	5
LF	1	157
LF	1	751
LF	1	151000
END	21	100574

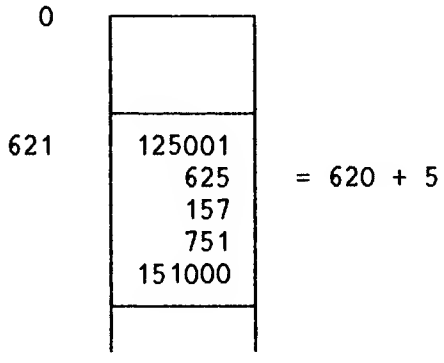
The contents of the control byte will form the control number. Control number 17 (mnemonic BEG) marks the beginning of the program. In FORTRAN, COBOL etc., control number 17 (BEG) is followed by control number 32 (LONG) which indicates that all S-groups contain six bytes instead of four. Control number 1 (LF) is followed by a P-group which is to be loaded unmodified, while control number 2 (LR) is followed by a P-group which contains an address relative to the beginning of the program, and which should therefore be modified. Control number 21 (END) is followed by a checksum.

Symbols (labels) are represented by S-groups where the six last bits are zero. (Note that in the example above, 125001 denotes the beginning of the program and is not a label.)

6.2 Relocation of Internal Addresses

Suppose that the load address is set to location 621 (either as a consequence of previous loading or by using the DEFINE command), and that we are going to load the example program we have looked at.

When the BRF-Linker begins loading, it reads control number 17 (BEG). The current location minus 1 is taken as the program's first address (also called the "program base"). In this case, the program base is 620. When loading, the program base is added to all P-groups which are preceded by the control number 2 (LR). The result is shown below.



6.3 Program Units

A program is composed of one main program and zero or more subprograms. A common name for main programs and subprograms is **program units**.

When a compiler compiles a program, each program unit is translated without any information about other program units. Therefore, the program units need not be compiled at the same time. Compilation of some program units separately from other program units is called separate compilation.

The address (or addresses) of a program unit where the execution begins is called the **entry point**. If the program unit is a main program, the entry point is called the **start address**. A word containing a reference to an entry point in another program unit is termed an **external reference**.

6.4 Separate Compilation

The object program consists of one or more BRF program units. The information necessary to link these together to an executable program, namely the entry points and the external references, is symbolic, and is placed in the S-groups. The meaning of the S-group is determined by the preceding control number in the following way:

Control Number	Mnemonic	Meaning
14	MAIN	Symbolic start address
15	LIBR	Library subprogram entry point
16	ENTR	Symbolic entry point
20	REF	Symbolic external reference

The object program units begin with control number 17 (BEG), end with control number 21 (END) and may contain one of the control numbers 14 (MAIN) or 16 (ENTR). A library subprogram has a control number 15 (LIBR) in addition to the 16 (ENTR). A library subprogram is loaded only when the LIBR symbol has been referenced by a REF group and is not already defined as a symbolic entry point. Library subprograms which are not needed are checked through to the END group.

If the BRF-Linker does not receive any other information, the program units are loaded consecutively, starting at a system-defined address. However, the program units may be loaded elsewhere by means of the control numbers:

10 (SFL) Start (continue) loading at the location in the P-group.

11 (AFL) Continue at the current location + the relative address in the P-groups.

12 (SRL) Continue at the current program base + the relative address in the P-group.

The main program and the subprograms may be read in an arbitrary sequence. If the program unit A refers to another program unit B, it does not matter which of them is loaded first. The (necessary) library subprograms are loaded last. But if the library subprogram A refers to another library subprogram B, then A must be loaded first, otherwise B will not be loaded.

6.5 Linking of Program Units

The BRF-Linker has a symbol table where each entry consists of three words for the symbol (the S-group) and one word (ADR) for the address.

ADR may have two different meanings:

1) If a symbolic entry point has been read, then ADR is the memory address of the entry point.

2) If only symbolic external references to a symbol have been read, then the ADR is a pointer to the last location at which the symbol was referenced. This location contains a pointer to the preceding reference to the same symbol, and so on. The first reference location contains the word 177777B to mark the end of this list. One bit in the table entry is used to discriminate between the two interpretations of ADR.

When a symbolic entry point is defined, any previous external references to this symbol will immediately be changed to the defined memory address of the symbol. This is done by following the list of references to the symbol described above.

6.6 FORTRAN COMMON Blocks

Some special BRF control numbers are used to ease the implementation of FORTRAN COMMON areas and data space allocation in general.

The memory area in which the BRF-Linker puts the program is a continuous area from a lower address up to the upper bound. The program units therefore normally grow upwards. For one-bank programs (but not for two-bank programs), COMMON blocks are allocated from the upper bound downwards. Thus the COMMON block address is found by subtracting the length from the upper bound and reducing the upper bound appropriately.

For two-bank programs, COMMON blocks are allocated from the present data load address upwards like all other data areas.

The COMMON block address must be known before the addresses referencing COMMON are loaded. Therefore the COMMON block address which uniquely specifies the maximum block length is defined by the first program unit using COMMON data. This explains the restriction that a COMMON block cannot be expanded by the succeeding program units.

The ASF group has the format:

<ASF><S-group><P-group>

where the S-group contains the name of the COMMON block, and the P-group contains the block length.

Data in COMMON is referenced by indirect addressing. Such addresses are followed by the control number 27 (ADS) which tells the BRF-Linker to add the COMMON block address.

The ADS-group has the format:

<ADS><S-group>

with the interpretation that the value of the S-group is added to the previously loaded address (P-group).

6.7 Fix-up Facilities

The BRF code is designed to allow single-pass, sequential transformation. This implies that the BRF-Linker must be able to fix words which have already been loaded. This is done by the four control numbers 4 (AFF), 5 (ARF), 6 (AFR), 7 (ARR) which all have two P-groups. The second P-group contains an address, and the first P-group has contents which will be added to that address. Both the address and the contents of the first P-group (which may be an address) may be relocated relative to the program base, and this therefore gives four possibilities.

6.8 Checksum

In order to detect read errors during loading, a checksum is placed behind each END control byte. Here, everything from the BEG control byte to the END control byte is added together, complemented and put in a P-group. The control bytes are regarded as eight bits, the P-group as sixteen bits, and the S-group as two or three sixteen bit numbers.

6.9 Description of the BRF Control Numbers

The legal control numbers are consecutive numbers starting at zero and are interpreted as commands to the BRF-Linker. They are listed in the following table together with their mnemonics and interpretation.

The terminology needs some explanation:

CLC is the current location counter. It contains the address where the next word is to be placed.
PB is the program base of the current program unit.
CDB is the COMMON data base (COMMON block address).
W1 and Wn are the contents of the first to the n'th P-group, respectively.

If "a" is an address or an address expression, then (a) is the content of this address. The expression X ⟶ (Y) means that the value X will replace the contents of Y, while X ⟶ ((Y)) means that the value X will be copied to the location having the address found in Y (indirect addressing).

BRF control numbers

Control Number (octal)	Mnemonic	No. of Words	Interpretation
0	FEED	0	Ignored
1	LF	1	W1→((CLC)),(CLC)+1→(CLC)
2	LR	1	W1+(PB)→((CLC)),(CLC)+1→(CLC)
3	LC	1	W1+(CDB)→((CLC)),(CLC)+1→(CLC)
4	AFF	2	W1+(W2)→(W2)
5	ARF	2	W1+(PB)+(W2)→(W2)
6	AFR	2	W1+(W2+(PB))→(W2+(PB))
7	ARR	2	W1+(PB)+(W2+(PB))→(W2+(PB))
10	SFL	1	W1→(CLC)
11	AFL	1	W1+(CLC)→(CLC), fill zeros
12	SRL	1	W1+(PB)→(CLC)
13		-	Not Used
14	MAIN	2(3)	Symbol in S-group will become the main entry
15	LIBR	2(3)	Conditional loading
16	ENTR	2(3)	Symbol in the S-group is assigned value of CLC
17	BEG	0	(CLC)→(PB) First control byte of a unit
20	REF	2(3)	Symbol in S-group is referenced in CLC
21	END	1	W1 contains the BRF-checksum
22	INHB	0	Warns that compilation errors have occurred
23	EOF	0	End of loading
24	LNF	1+W1	W2,W3,...,Wn→(CLC),...,(CLC+W1-1)

BRF control numbers - continued

Control Number (octal)	Mnemonic	No. of Words	Interpretation
25	RT	1	W1 contains real time priority
26	ASF	3(4)	<symbol><number> Defines common length. Value of symbol in loader table = common start address.
27	ADS	2(3)	<symbol>+(CLC-1)→(CLC-1) Adds common address
30	MSG	1+W1	W1 contains length of message in words
31		-	Not used
32	LONG	0	Flags a six-byte S-group
33		-	Not used
34	INL	2	W2→(W1+(PB))
35	DBL	3	Wi→(W1+(PB)+i-2) (i = 2 to 3)
36	RLL	4	Wi→(W1+(PB)+i-2) (i = 2 to 4)
37	CXL	7	Wi→(W1+(PB)+i-2) (i = 2 to 7)
40 *	INC	4(5)	W5→(W4 + ADR)
41 *	DBC	5(6)	Wi→(W4 + ADR + i-5) (i = 5 to 6)
42 *	RLC	6(7)	Wi→(W4 + ADR + i-5) (i = 5 to 7)
43 *	CXC	9(10)	Wi→(W4 + ADR + i-5) (i = 5 to 10)
44	BYL	2	W2(bit 0-7)→(W1+(PB))(bit 0-7) if W2 bit 15=0 W2(bit 0-7)→(W1+(PB))(bit 8-15) if W2 bit 15=1
45 *	BYC	5	W5(bit 0-7)→(W4 + ADR)(bit 0-7) if W5 bit 15=0 W5(bit 0-7)→(W4 + ADR)(bit 8-15) if W5 bit 15=1
46	NWL	1	W1 contains line number. (Not in use.)
47	DBG	0	Indicates start/stop of Debug information

BRF control numbers - continued

Control Number (octal)	Mnemonic	No. of Words	Interpretation
50	PMO	0	Indicates start of program bank mode
51	DMO	0	Indicates start of data bank mode
52	LRP	1	Same as LR but PB of program bank
53	LRD	1	Same as LR but PB of data bank
54	DIC	-	Dictionary table follows. Each element contains name (3 words) and byte pointer (2 words). End of table marked by -1.

* The W1, W2, and W3 contain a common block name. At load time this symbol must be defined. Its value is referred to as ADR.

APPENDIXES

A. COMMAND SUMMARY

In this appendix the various commands of the BRF-Linker are briefly described.

The BRF-Linker is controlled from the terminal by the following command words. They may be abbreviated provided no ambiguity results. The parameters, if any, are separated by a space or a comma.

Brl: APPEND-BRF <source file>,<destination file>,<after unit>

Insert all BRF units in the source file into the destination file after the specified unit. If no unit is specified, append the units from the source file at the end of the destination file.

Brl: COPY-PROGFILE <source file>,<destination file>
[,<Include Debug? YES/NO>]
[,<Include Link Information? YES/NO>]

The <source file> is the name of the file to copy from, <destination file> is the name of the file to copy to. The default file type is :PROG for both files. The parameter <Include debug? YES/NO> gives the user an opportunity to include debug information during copying. Answer YES to include it or NO to delete it. The default answer is NO. For files using the multisegment system, link information can be deleted while copying. The parameter <Include Link Information?> gives the user an opportunity to include multisegment link information. The default answer is NO.

Brl: DEBUG-MODE <ON/OFF>

Debug information on BRF files can be accepted or ignored. Default parameter is ON.

Brl: DEFINE <symbol>,<address>,<P/D>

The symbol will be entered into the BRF-Linker's symbol table. Its value and mode will be equal to what is specified. Default mode is P (program mode).

Brl: DEFINE <symbol>,?,<P/D>

If defined, the value of the symbol specified will be printed on the terminal.

Brl: DEFINE <#PCLC/#DCLC>,<address>

Subsequent loading in the specified bank will start from the address specified.

Brl: DELETE-BRF <file name>,<first unit>,<last unit>

Delete a sequence of BRF units from the specified file starting with the <first unit> and delete the following units up to and including the <last unit>.

Brl: EXIT

Control is returned to SINTRAN III.

Brl: FETCH-BRF <source file>,<destination file>,<first unit>,
<last unit>

Fetch a sequence of BRF units from the source file, starting with the <first unit> and taking all following units up to and including the <last unit>, and append them at the end of the <destination file>.

Brl: HELP [<command>]

List the available loader commands matching <command> on the terminal. If no command name is specified, all commands will be listed.

Brl: INSERT-BRF-MESSAGE <file name>,<before unit>,<message>

Insert a message before the specified unit on a given file. The message will be printed on the terminal when the file is loaded.

Brl: LIBRARY-MODE <ON/OFF>

Library files can be loaded in library mode or normal (non-library) mode. The default value is ON.

Brl: LINK-TO <file-1>,[<file-2>,....,<file-n>]

Perform multisegment linking between the program file (as specified in the **PROGRAM-FILE** command) and the files specified in this command. The default file type is :PROG.

Brl: LIST-BRF-CODE <file name>,<first unit>,<last unit>,
<output file>

List information from a sequence of BRF units in the specified source file on the <output file>, starting with the <first unit> and ending with the <last unit>.

Brl: LIST-BRF-ENTRIES <file name>,<output file>

List all defined symbols in all BRF units in the specified source file on the specified output file.

Brl: LIST-ENTRIES-DEFINED ,,,

All defined symbols in the BRF-Linker's symbol table (in both program code and data banks) and the current address/value will be printed on the terminal.

Brl: LIST-ENTRIES-UNDEFINED ,,,

This command is similar to **LIST-ENTRIES-DEFINED,,,** except that undefined symbols are printed.

Brl: LOAD <file name>[,<file name>...]

The file(s) specified will be loaded until the end-of-file marker is encountered. The default file type is :BRF.

Brl: LOOK-AT-DATA <address>

Used to inspect and modify data locations.

Brl: LOOK-AT-PROGRAM <address>

Used to inspect and modify program locations.

Brl: OUTPUT-FILE <file name>

This command is used to specify that output is to be written to the specified file instead of the terminal. Output from the following commands: **LIST-ENTRIES-DEFINED**, **LIST-ENTRIES-UNDEFINED**, **LINK-TO**, **PROGRAM-INFORMATION**, **LIST-BRF-CODE** and **LIST-BRF-ENTRIES** will be written to the file specified. The default file type is :SYMB. To reset output to the terminal, give the command **OUTPUT-FILE** with no file name.

Brl: OVERLAY <level>,<entry name 1>[,...,<entry name n>]

This command specifies that the next overlay link is to be generated. The <level> is the overlay level. The parameters <entry name 1> to <entry name n> are the names of the subprograms called from the previous level. The root link is level 0. A level must always be specified when linking overlays.

Brl: PATCH-PROGFILE-NAME <file name>,<new name>

This command is used to change the name used in the **PROGRAM-FILE** command when the program file specified by <file name> was written. If the SINTRAN III command **RENAME-FILE** is used to rename a file, the **PATCH-PROGFILE-NAME** command can be used to change the file name written on the file. Note that this command will not change the SINTRAN III file name.

Brl: PREPARE-BRF-LIBRARY-FILE <source file>

Generate an index table of all BRF units in the <source file> and insert this index table as a new unit at the very beginning of the file.

Brl: PROGRAM-FILE <file name>[/<segment name>][,<W>]

The output from the BRF-Linker will be loaded onto the file specified. The default file type is :PROG. The /<segment name> parameter is used to specify the segment name in multisegment mode, and the <W> parameter is used to indicate that only the program inspection commands are to be used on an existing program file.

Brl: PROGRAM-INFORMATION <file name>
[,<Dump Link Information?YES/NO>,<output file>]

Information concerning the specified program file will be listed. The default file type is :PROG. The two last parameters are only valid for multisegment program files. The default file type for the output file is :SYMB.

Brl: REFERENCE <symbol>,<address>,<P/D>

This command is used to insert or refer to an undefined symbol in the BRF-Linker's symbol table. The following rules apply:

1) If the symbol is not present in the symbol table, the value -1 will be put into the specified address and this address will be referenced in the table. The specified octal address must be an unused memory address, otherwise the information stored there previously will be written over. If no address is given, then the symbol will be treated as a referenced symbol only.
2) If the symbol is present, but already referenced (undefined), the address specified will be linked into the reference chain.
3) If the symbol is defined, its value will be put into the address specified.
4) The default bank is P (program bank).

Brl: REMOVE <symbol>,<P/D>

If present, this symbol will be removed from the BRF-Linker's symbol table.

Brl: RENAME <old symbol>,<new symbol>

This command is used to give the specified symbol a new name. Subsequent references to the <old symbol> will be assumed to be references to another symbol with the old name.

Brl: RENAME-BRF <file name>,<old symbol>,<new symbol>

This command is used to change the name of a symbol (<old symbol>) in a specified BRF file.

Brl: REPLACE-BRF <source file>,<destination file>

Replace BRF units on the destination file with units from the source file. Units found only in the destination file will not be changed, whereas units only found in the source file will be ignored, giving a warning message.

Brl: RESTART <address>

or

Brl: RESTART <symbol>

To set the restart address (the address that the program starts executing from when you type @**CONTINUE** at your terminal) of the program file specified in **PROGRAM-FILE** command. The <symbol> must be a defined entry in the program area. The default restart address will be equal to the main start address.

Brl: RUN

This command leaves the BRF-Linker and then starts executing the program file opened with the **PROGRAM-FILE** command at the beginning of the loading session.

B. ERROR MESSAGES

When an error occurs during a loading session, the BRF-Linker types the text **Brl message:** followed by an error message on the terminal or output device. The various error messages are listed below in alphabetical order.

In addition to these messages, some of the file system error messages may appear on your terminal.

AMBIGUOUS COMMAND

The last command name has been abbreviated and is not unique.

CHECKSUM ERROR

The BRF file contents have been corrupted as a result of hardware or software errors occurring during reading or writing.

COMMON BLOCK EXHAUST AVAILABLE SPACE

The common block size is too large for the remaining free area.

COMMON BLOCK EXPANDED

The length of a previously defined common block has been declared to be larger in a subsequently loaded program.

COMPILER SYSTEM ERROR

Erroneous use of generated labels in the compiler.

DATA SPACE EXCEEDED

The current load address of the data has reached the maximum limit of 64 pages.

DEBUG TABLE FULL

The current address for debug information has reached the absolute upper limit of the free area.

FILE DOES NOT CONTAIN BRF-CODE

Non-interpretable information has appeared on the BRF file.

xxxxx FIRST UNIT IS NOT PRIOR TO LAST UNIT

The BRF unit xxxxx is not prior to the <last unit> specified.

ILLEGAL OVERLAY LEVEL

The overlay level must not be increased by more than 1 from the last OVERLAY command; the first time it must be 0.

ILLEGAL SEQUENCE OF OVERLAYS

An overlay has referenced a symbol which is not in its path, nor in any links immediately below it.

INVALID ADDRESS

An address specified in the last command is not a valid address.

xxxxx INVALID ADDRESS OR NOT DEFINED SYMBOL

The symbol or address xxxxx specified in the last command is not a valid address or a defined symbol.

INVALID COMMAND

The last command name is unknown.

INSUFFICIENT BRF-UNIT, SYNTAX ERRORS

Errors have occurred during the compilation process.

MIXED ONE/TWO BANK ROUTINES

Routines compiled with the compiler command **SEPARATE-DATA OFF** may not be mixed with routines compiled with **SEPARATE-DATA ON**. There is an exception in the case of routines written in MAC and NPL.

NEW CHECKSUM GENERATED

Using the command **RENAME** to rename a symbol will cause a checksum error. To overcome this, a new checksum is generated and written to the BRF file. Note that this message does not necessarily indicate an error.

NO MAIN ENTRY

The user is trying to start a program having no main program module.

NO PROGRAM-FILE SPECIFIED

The command **PROGRAM-FILE** must be used before any files can be loaded.

NO SUCH FILE

The file name specified in the command is not a legal file name.

xxxxx NOT FOUND IN DESTINATION FILE

The BRF unit xxxxx is not a unit (entry) in the destination file.

xxxxx NOT FOUND IN SOURCE FILE

The BRF unit xxxxx is not a unit (entry) in the source file.

OVERLAPPING DATA IN LINKED SEGMENTS

The local data corresponding to each code segment must be loaded into different areas in the data segment.

PROGRAM SPACE EXCEEDED

The current load address of the program area has reached the maximum limit of 64 pages.

PROGRAM SYSTEM TOO LARGE

During overlay loading, the overlaid program system has become too large for the BRF-Linker to handle.

REDEFINITION. LAST APPLIES xxxx yyyy.

The symbol xxxx being defined (either by loading a file or by the **DEFINE** command) has already been assigned an octal value yyyy. The first value defined for the symbol is kept.

REFERENCED ELSEWHERE THAN CURRENT OR PREVIOUS LEVEL

During overlay loading, references should only be to the current or the next level.

ROOT-SEGMENT NOT INITIATED (OVERLAY 0)

In overlay loading, the overlay system must be initiated by the command **OVERLAY 0**,,.

SEGMENT-ROUTINE NOT LOADED

In multisegment loading, the routine for segment switching is not loaded. The library must be loaded.

xxxxx SYMBOL NOT FOUND

The symbol xxxxx is not found in the symbol table.

TOO LONG NAME. WILL BE TRUNCATED

The name is too long and will be truncated to a maximum of 15 characters.

UNDEFINED COMMON LABEL

Undefined common block in program.

UNDEFINED ENTRIES

Undefined entries in loaded program.

Index

Absolute program . . . . . 2, 4.
Absolute program files . . . . . 2.
Address symbolic start . . . . . 44.
AMBIGUOUS COMMAND . . . . . 59.
Angular brackets . . . . . 1.
APPEND-BRF command . . . . . 37, 53.
Backward reference in overlay systems . . . . . 15.
Binary
  program . . . . . 41.
  Relocatable Format . . . . . 1, 41.
BRF . . . . . 1, 41.
  code . . . . . 42.
  control number . . . . . 42, 46.
  P-group . . . . . 42.
  S-group . . . . . 42.
BRF-Linker
  commands . . . . . 1, 53.
  input . . . . . 4.
  modes . . . . . 2.
BRF Control
  byte . . . . . 42.
  information . . . . . 42.
  number . . . . . 46.
BRF Symbolic information . . . . . 42.
Brl message . . . . . 59.
Carriage return . . . . . 1.
Checksum . . . . . 46.
CHECKSUM ERROR . . . . . 59.
Comma . . . . . 1.
Command
  APPEND-BRF . . . . . 37, 53.
  COPY-PROGFILE . . . . . 11, 53.
  DEBUG-MODE . . . . . 4, 53.
  DEFINE . . . . . 7, 8, 25, 53.
  DELETE-BRF . . . . . 38, 54.
  DUMP-PROGRAM-REENTRANT . . . . . 26.
  EXIT . . . . . 5, 25, 54.
  FETCH-BRF . . . . . 38, 54.
  HELP . . . . . 12, 54.
  INSERT-BRF-MESSAGE . . . . . 38, 54.
  LIBRARY-MODE . . . . . 4, 54.
  LINK-TO . . . . . 24, 54.
  LIST-BRF-CODE . . . . . 39, 54.
  LIST-BRF-ENTRIES . . . . . 37, 55.
  LIST-ENTRIES-DEFINED . . . . . 7, 55.
  LIST-ENTRIES-UNDEFINED . . . . . 7, 55.
  LOAD . . . . . 4, 55.
  LOAD-REENTRANT-SEGMENT . . . . . 26.
  LOOK-AT-DATA . . . . . 35, 55.
  LOOK-AT-PROGRAM . . . . . 35, 55.
  Multisegment PROGRAM-FILE . . . . . 24.
  OUTPUT-FILE . . . . . 7, 25, 55.

OVERLAY . . . . . 16, 55.
PATCH-PROGFILE-NAME . . . . . 10, 56.
PREPARE-BRF-LIBRARY-FILE . . . . . 38, 56.
PROGRAM-FILE . . . . . 4, 24, 35, 56.
PROGRAM-INFORMATION . . . . . 10, 56.
REFERENCE . . . . . 8, 56.
REMOVE . . . . . 8, 57.
RENAME . . . . . 8, 57.
RENAME-BRF . . . . . 39, 57.
REPLACE-BRF . . . . . 38, 57.
RESTART . . . . . 8, 57.
RUN . . . . . 5, 57.
Command abbreviation . . . . . 1.
Command format . . . . . 1, 53.
Command summary . . . . . 53.
Commands for loading overlays . . . . . 16.
COMMON
addressing . . . . . 45.
block . . . . . 46.
declaration . . . . . 45.
expansion . . . . . 45.
length . . . . . 45.
COMMON block address . . . . . 46.
COMMON BLOCK EXHAUST AVAILABLE SPACE . . . . . 59.
COMMON BLOCK EXPANDED . . . . . 59.
COMMON blocks and Multisegment linking . . . . . 25.
COMPILER SYSTEM ERROR . . . . . 59.
Control
byte in BRF . . . . . 42.
character . . . . . 1.
information in BRF . . . . . 42.
numbers in BRF . . . . . 42.
COPY-PROGFILE command . . . . . 11, 53.
Copying program files . . . . . 11.
Current location counter . . . . . 46.
Data . . . . . 8, 53.
Program . . . . . 8, 53.
Data
inspection . . . . . 35, 55.
modification . . . . . 35, 55.
DATA SPACE EXCEEDED . . . . . 59.
Debug information . . . . . 4, 53.
DEBUG TABLE FULL . . . . . 59.
DEBUG-MODE command . . . . . 4, 53.
Debugger and Overlays . . . . . 15.
Decimal number . . . . . 1.
Default
file type . . . . . 53.
LIBRARY-MODE . . . . . 4.
load-file type . . . . . 55.
restart address . . . . . 8.
Default file type . . . . . 53.

Default load-file type . . . . . 55.
Default restart address . . . . . 8.
Default values for missing parameter . . . . . 1.
DEFINE command . . . . . 7, 8, 25, 53.
Defined symbols . . . . . 6.
DELETE-BRF command . . . . . 38, 54.
Demand paging . . . . . 3.
Dependent links in overlay systems . . . . . 13.
Design of an overlay system . . . . . 15.
DUMP-PROGRAM-REENTRANT command . . . . . 26.
Dumping . . . . . 24.
  Multisegment program . . . . . 22.
  Overlay program . . . . . 16.
Editing commands . . . . . 37, 53.
Entry point . . . . . 43.
Error
  AMBIGUOUS COMMAND . . . . . 59.
  CHECKSUM ERROR . . . . . 59.
  COMMON BLOCK EXHAUST AVAILABLE SPACE . . . . . 59.
  COMMON BLOCK EXPANDED . . . . . 59.
  COMPILER SYSTEM ERROR . . . . . 59.
  DATA SPACE EXCEEDED . . . . . 59.
  DEBUG TABLE FULL . . . . . 59.
  FILE DOES NOT CONTAIN BRF-CODE . . . . . 59.
  FIRST UNIT IS NOT PRIOR TO LAST UNIT . . . . . 60.
  ILLEGAL OVERLAY LEVEL . . . . . 60.
  ILLEGAL SEQUENCE OF OVERLAYS . . . . . 60.
  INSUFFICIENT BRF-UNIT, SYNTAX ERRORS . . . . . 60.
  INVALID ADDRESS . . . . . 60.
  INVALID ADDRESS OR NOT DEFINED SYMBOL . . . . . 60.
  INVALID COMMAND . . . . . 60.
  MIXED ONE/TWO BANK ROUTINES . . . . . 60.
  NEW CHECKSUM GENERATED . . . . . 60.
  NO MAIN ENTRY . . . . . 60.
  NO PROGRAM-FILE SPECIFIED . . . . . 61.
  NO SUCH FILE . . . . . 61.
  NOT FOUND IN DESTINATION FILE . . . . . 61.
  NOT FOUND IN SOURCE FILE . . . . . 61.
  OVERLAPPING DATA IN LINKED SEGMENTS . . . . . 61.
  PROGRAM SPACE EXCEEDED . . . . . 61.
  PROGRAM SYSTEM TOO LARGE . . . . . 61.
  REDEFINITION. LAST APPLIES xxxx yyyy. . . . . 61.
  REFERENCED ELSEWHERE THAN CURRENT / PREVIOUS LEVEL 61.
  ROOT-SEGMENT NOT INITIATED OVERLAY 0 . . . . . 61.
  SEGMENT-ROUTINE NOT LOADED . . . . . 62.
  SYMBOL NOT FOUND . . . . . 62.
  TOO LONG NAME. WILL BE TRUNCATED . . . . . 62.
  UNDEFINED COMMON LABEL . . . . . 62.
  UNDEFINED ENTRIES . . . . . 62.
Executable program . . . . . 2, 4.
Executable program files . . . . . 2.
Executing overlay programs . . . . . 16.

Execution time for overlay systems . . . . . 3.
EXIT command . . . . . 5, 25, 54.
Extended path in overlay systems . . . . . 14.
External reference . . . . . 43.
FETCH-BRF command . . . . . 38, 54.
FILE DOES NOT CONTAIN BRF-CODE . . . . . 59.
FIRST UNIT IS NOT PRIOR TO LAST UNIT . . . . . 60.
Fix-up . . . . . 46.
FORTRAN COMMON . . . . . 45.
FORTRAN COMMON and
  Multisegment linking . . . . . 25.
  Overlays . . . . . 15.
Forward reference in overlay systems . . . . . 15.
HELP command . . . . . 12, 54.
ILLEGAL OVERLAY LEVEL . . . . . 60.
ILLEGAL SEQUENCE OF OVERLAYS . . . . . 60.
Independent links in overlay systems . . . . . 13.
INSERT-BRF-MESSAGE command . . . . . 38, 54.
INSUFFICIENT BRF-UNIT, SYNTAX ERRORS . . . . . 60.
Intermodule references . . . . . 1.
  Multisegment program . . . . . 22.
INVALID ADDRESS . . . . . 60.
INVALID ADDRESS OR NOT DEFINED SYMBOL . . . . . 60.
INVALID COMMAND . . . . . 60.
Label . . . . . 1.
Library
  files . . . . . 4.
  object programs . . . . . 44.
  subprograms . . . . . 44.
LIBRARY-MODE command . . . . . 4, 54.
Library object programs . . . . . 44.
Library subprogram entry point . . . . . 44.
Link path in overlay . . . . . 14.
LINK-TO command . . . . . 24, 54.
Links
  dependent . . . . . 13.
  independent . . . . . 13.
LIST-BRF-CODE command . . . . . 39, 54.
LIST-BRF-ENTRIES command . . . . . 37, 55.
LIST-ENTRIES-DEFINED command . . . . . 7, 55.
LIST-ENTRIES-UNDEFINED command . . . . . 7, 55.
LOAD
  address . . . . . 6.
  command . . . . . 4, 55.
LOAD-REENTRANT-SEGMENT command . . . . . 26.
Loading . . . . . 2.
  errors . . . . . 46.
  library files . . . . . 4.
Loading library files . . . . . 4.
Location counter, current . . . . . 46.
LOOK-AT-DATA command . . . . . 35, 55.
LOOK-AT-PROGRAM command . . . . . 35, 55.

Missing parameter . . . . . 1.
MIXED ONE/TWO BANK ROUTINES . . . . . 60.
Monitor Call RFILE . . . . . 3.
Multilevel overlay system . . . . . 13.
Multisegment
  linking . . . . . 22, 54.
  linking command . . . . . 24.
  mode . . . . . 3.
  PROGRAM-FILE command . . . . . 24.
  restart address . . . . . 25.
  start address . . . . . 25.
Multisegment and
  FORTRAN COMMON . . . . . 25.
  Overlay . . . . . 22.
Multisegment file information . . . . . 10.
Multisegment linking command . . . . . 24.
Multisegment link information stripping . . . . . 53.
Multisegment PROGRAM-FILE command . . . . . 24.
Named segments two-bank . . . . . 3.
NEW CHECKSUM GENERATED . . . . . 60.
Normal mode . . . . . 3.
NO MAIN ENTRY . . . . . 60.
NO PROGRAM-FILE SPECIFIED . . . . . 61.
NO SUCH FILE . . . . . 61.
NOT FOUND IN DESTINATION FILE . . . . . 61.
NOT FOUND IN SOURCE FILE . . . . . 61.
Object program . . . . . 44.
Object program unit . . . . . 44.
Octal number . . . . . 1.
One-bank COMMON . . . . . 45.
OUTPUT-FILE command . . . . . 7, 25, 55.
OVERLAPPING DATA IN LINKED SEGMENTS . . . . . 61.
Overlay . . . . . 13.
  command . . . . . 16, 55.
  debugging . . . . . 15.
  execution time . . . . . 3.
  linking . . . . . 55.
  loading . . . . . 16, 55.
  mode . . . . . 3.
  program execution . . . . . 16.
  structure . . . . . 13.
  system . . . . . 3, 13.
Overlay links with extended paths . . . . . 14.
Overlay loading commands . . . . . 16.
Overlay program execution . . . . . 16.
Overlay program information . . . . . 10.
Overlay system design . . . . . 15.
Overlay systems and
  backward reference . . . . . 15.
  dependent links . . . . . 13.
  forward reference . . . . . 15.
  independent links . . . . . 13.

Overlays and
FORTRAN COMMON . . . . . 15.
Multisegment . . . . . 22.
Symbolic Debugger . . . . . 13.
P-group in BRF . . . . . 42.
Page . . . . . 2.
Parameter delimiter . . . . . 1.
PATCH-PROGFILE-NAME command . . . . . 10, 56.
Path loading in overlay systems . . . . . 14.
PREPARE-BRF-LIBRARY-FILE command . . . . . 38, 56.
Program . . . . . 41.
base . . . . . 43, 46.
file . . . . . 4.
information . . . . . 10.
inspection . . . . . 35, 55.
modification . . . . . 35, 55.
relocatable . . . . . 2.
unit . . . . . 43.
PROGRAM-FILE
command . . . . . 4, 24, 35, 56.
command, inspection mode . . . . . 35.
PROGRAM-INFORMATION command . . . . . 10, 56.
PROGRAM SPACE EXCEEDED . . . . . 61.
PROGRAM SYSTEM TOO LARGE . . . . . 61.
Programmed information . . . . . 42.
RECOVER SINTRAN III command . . . . . 5.
REDEFINITION. LAST APPLIES xxxx yyyy. . . . . 61.
Reentrant named segments . . . . . 21.
Reentrant program dumping . . . . . 26.
REFERENCE command . . . . . 8, 56.
REFERENCED ELSEWHERE THAN CURRENT OR PREVIOUS LEVEL . 61.
Relocatable program . . . . . 2, 41.
Relocatable program file . . . . . 2.
REMOVE command . . . . . 8, 57.
RENAME command . . . . . 8, 57.
RENAME-BRF command . . . . . 39, 57.
RENAME-FILE SINTRAN III command . . . . . 11.
Renaming symbols . . . . . 8.
REPLACE-BRF command . . . . . 38, 57.
Restart
address . . . . . 8.
address multisegment . . . . . 25.
command . . . . . 8, 57.
Root link . . . . . 13.
ROOT-SEGMENT NOT INITIATED OVERLAY 0 . . . . . 61.
RUN command . . . . . 5, 57.
S-group . . . . . 44.
S-groups in BRF . . . . . 42.
SEGMENT-ROUTINE NOT LOADED . . . . . 62.
Segments . . . . . 21.
background . . . . . 21.
Separate
assembly . . . . . 43.

compilation . . . . . 43.
Signed decimal number . . . . . 1.
Space . . . . . 1.
Square brackets . . . . . 1.
Start
address . . . . . 43.
address multisegment . . . . . 25.
Stripping multisegment link information . . . . . 53.
Subsystem . . . . . 2.
Switching times . . . . . 3.
Symbol . . . . . 1, 6.
entering . . . . . 53.
entry . . . . . 1.
length . . . . . 6.
table . . . . . 1, 6, 44.
Symbolic
Debugger . . . . . 3, 5.
entry point . . . . . 44, 45.
external reference . . . . . 44, 45.
information BRF . . . . . 42.
Symbolic Debugger and Overlays . . . . . 13, 15.
Symbolic start address . . . . . 44.
Symbol BRF . . . . . 41.
SYMBOL NOT FOUND . . . . . 62.
TOO LONG NAME. WILL BE TRUNCATED . . . . . 62.
Two-bank
COMMON . . . . . 45.
named segments . . . . . 3.
Undefined
symbol . . . . . 7.
symbol entry . . . . . 1.
UNDEFINED COMMON LABEL . . . . . 62.
UNDEFINED ENTRIES . . . . . 62.
User program execution . . . . . 5.
Word . . . . . 2.

********** **SEND US YOUR COMMENTS!!!** **********

Are you frustrated because of unclear information in this manual? Do you have trouble finding things? Why don't you join the Reader's Club and send us a note? You will receive a membership card — and an answer to your comments.

Please let us know if you
- find errors
- cannot understand information
- cannot find information
- find needless information

Do you think we could improve the manual by rearranging the contents? You could also tell us if you like the manual!

********** **HELP YOURSELF BY HELPING US!!** **********

Manual name: BRF–LINKER User Manual

Manual number: ND–60.196.01

What problems do you have? (use extra pages if needed) _____

_____

_____

_____

_____

Do you have suggestions for improving this manual ? _____

_____

_____

_____

_____

_____

Your name: _____ Date: _____

Company: _____ Position: _____

Address: _____

_____

What are you using this manual for ? _____

_____

NOTE!
This form is primarily for documentation errors. Software and system errors should be reported on Customer System Reports.

Send to:
Norsk Data A.S
Documentation Department
P.O. Box 25, Bogerud
Oslo 6, Norway

→

Norsk Data's answer will be found on reverse side

Answer from Norsk Data _____

_____

_____

_____

_____

_____

_____

_____

_____

_____

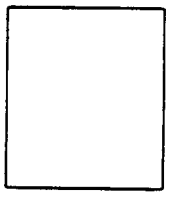
_____

_____

_____

_____

Answered by_____ Date _____



Norsk Data A.S
Documentation Department
P.O. Box 25, Bogerud
0621 Oslo6, Norway